BlackBerry Java Application Multimedia

Version: 5.0

Development Guide

BlackBerry

Published: 2010-12-8
SWD-741609-1208103256-001

Contents

Working with multimedia in a BlackBerry device application

1

Creating a BlackBerry device application that plays media

You can create a BlackBerry® device application that plays media in the BlackBerry® Browser or in the Media application on a BlackBerry device. A BlackBerry device media application can also be created to play audio, video, and binary SVG content. A BlackBerry device application can also be created to record audio and video, and send audio to a Bluetooth® enabled headset.

The Content Handler API, is also used to provide an execution model for remotely invoking non-core BlackBerry device applications. CHAPI is an ideal mechanism for setting invocation parameters for non-core BlackBerry device applications. Using CHAPI, you can invoke applications by providing either a URL, content type, or content ID when using one of the constructors available in javax.microedition.content.Invocation class.

Note: CHAPI will attempt to retrieve an invocation response unless you explicitly specify that no response is necessary.

You can create a BlackBerry device application to play media that uses the `Player` interface and the `javax.microedition.media` package. The `Player` interface has been implemented by RIM. It provides the methods needed to manage the different states of a BlackBerry device application that plays media and control the playback of media files.

A `Player` object is first constructed in an UNREALIZED state. While in this state the object is incapable of doing anything for lack of information and resources.

Invoking `realize()` will move the Player into a REALIZED state. This enables the Player to locate information to get the basic resources to play a media file.

Invoking `prefetch()` will move the Player into a PREFETCHED state. In this state the Player obtains necessary resources, and then "fetches" some of the media file to immediately start playback as soon as the Player has been started.

Invoking `start()` will move the Player into a STARTED start where the Player starts immediate playback of media.

It is possible to invoke `start()` without invoking `realize()` and `prefetch()`. The `start()` method will invoke `prefetch(0)`, which invokes `realize()` before media playback can be started. The simplicity of using `start()` in this manner may be offset through increased delays starting media playback.

For more information about media types that the BlackBerry device supports, read the knowledge base article KB05482 at www.blackberry.com/btsc

Create a BlackBerry device application that plays media

1. Import the required classes.

```
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
import java.io.IOException;
```

2. Retrieve a `Player` object from the `Manager` class by invoking the `createPlayer()` method.
3. Invoke `Player.realize()` to prepare the BlackBerry® device application to obtain the resources it requires.
4. Invoke `Player.prefetch()` to let the BlackBerry device application perform actions that must occur before the BlackBerry device application can play media.
5. Invoke `Player.start()` to start playing media.

```
Player p = Manager.createPlayer("http://www.test.rim.net/abc.wav");
/*
 * Best practice is to invoke realize(), then prefetch(), then start().
 * Following this sequence reduces delays in starting media playback.
 *
 * Invoking start() as shown below will cause start() to invoke  prefetch(0),
 * which invokes realize() before media playback is started.
 */
 p.start();
```

6. Invoke `Player.stop()` to stop the playback of media.
7. Invoke `Player.close()` to place the `Player` in a `CLOSED` state.

Code Sample: Playing media from a URI

```
private void initVideo(String url) {
    try
    {
        _player = javax.microedition.media.Manager.createPlayer(url);
        _player.realize();
        _vc = (VideoControl) _player.getControl("VideoControl");
        if (_vc != null)
        {
            _videoField = (Field) _vc.initDisplayMode(
     VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field");
            _vc.setVisible(true);
        }
    } catch(MediaException pe) {
        System.out.println(pe.toString());
    } catch (IOException ioe) {
        System.out.println(ioe.toString());
    }
}
```

Code sample: Playing a sequence of tones

`ToneControl` is used to play a user-defined sequence of notes at a specific duration and tempo on a BlackBerry® device.

```
// "Mary Had A Little Lamb" has "ABAC" structure
// Use block to repeat "A" section
// Tempos ranging from 20 to 508 beats per minute are divided by 4
// to create a tempo modifier range of 5 to 127.
byte tempo_mod = 30; // 120 bpm
// Note duration  ranges from 128 (1/2 note) to 0 (128th of a note)
// with a default resolution of 64.
byte duration  =  8; // Note length 8 (quaver) = 1/8th of a note duration
// Notes are determined from ToneControl.C4 (Middle C),
// which has a value of 60 and a frequency of 261.6 Hz.
byte C4 = ToneControl.C4; // C note value = 60 (middle C)
byte D4 = (byte)(C4 + 2); // D note value = 62 (a whole step)
byte E4 = (byte)(C4 + 4); // E note value = 64 (a major third)
byte G4 = (byte)(C4 + 7); // G note value = 67 (a fifth)
byte rest = ToneControl.SILENCE; // rest
byte[] mySequence = {
    ToneControl.VERSION, 1, // version 1
    ToneControl.TEMPO, tempo_mod,
    //
    // Start define "A" section
    ToneControl.BLOCK_START, 0,
    //
    // Content of "A" section
    E4, duration, D4, duration, C4, duration, E4, duration,
    E4, duration, E4, duration, E4, duration, rest, duration,
    //
    // End define "A" section
    ToneControl.BLOCK_END, 0, // end of block number 0
    //
    // Play "A" section
    ToneControl.PLAY_BLOCK, 0,
    //
    // Play "B" section
    D4, duration, D4, duration, D4, duration, rest, duration,
    E4, duration, G4, duration, G4, duration, rest, duration,
    //
    // Repeat "A" section
    ToneControl.PLAY_BLOCK, 0,
    //
    // Play "C" section
    D4, duration, D4, duration, E4, duration, D4, duration, C4, duration
};
try{
    Player p = Manager.createPlayer(Manager.TONE_DEVICE_LOCATOR);
    p.realize();
    ToneControl c = (ToneControl)p.getControl("ToneControl");
```

```
    c.setSequence(mySequence);
    p.start();
} catch (IOException ioe) { }
} catch (MediaException me) { }
```

Create a BlackBerry device application that plays media from an input stream

1. Import the required classes.

```
import java.lang.Class;
import javax.microedition.media.Player;
import javax.microedition.media.Manager;
import javax.microedition.rms.RecordStore;
import java.io.InputStream;
import java.io.ByteArrayInputStream;
import net.rim.device.api.media.protocol.ByteArrayInputStreamDataSource;
```

2. Create a `ByteArrayInputStream` object. In this example, `getResourceAsStream(String)` is invoked to retrieve the media file from the resource file in an absolute location.

```
ByteArrayInputStream stream = (ByteArrayInputStream)this.getClass
().getResourceAsStream("/abc.mp3");
```

3. Create a seekable `SourceStream` by creating a `ByteArrayInputStreamDataSource` by passing the `ByteArrayInputStream` object.

```
ByteArrayInputStreamDataSource source = new ByteArrayInputStreamDataSource
(stream, "audio/mpeg");
```

4. Retrieve a `Player` object by invoking `Manager.createPlayer()` and then `start()` media playback.

```
Player p = Manager.createPlayer(source);
p.start();
```

5. You could also use a non-seekable method to play media from an `InputStream` as shown in the following code example.

```
InputStream stream = (InputStream)this.getClass().getResourceAsStream("/abc.mp3");
Player p = Manager.createPlayer(stream, "audio/mpeg");
p.start();
```

Create a BlackBerry device application that plays streaming media

The BlackBerry® devices with BlackBerry® Device Software version 4.3.0 or later that supports EVDO will support RTSP functionality

1. Import the required classes.

```
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
```

2. Invoke `Manager.createPlayer(String)` to stream media by passing a parameter that represents an RTSP locator.

```
Player p = Manager.createPlayer("rtsp://streaming.rim
    .com/streaming_video.3gp");
```

Play a video in a UI field in a BlackBerry device application

1. Import the required classes and interfaces.

```
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

import java.io.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
```

2. Create the application framework by extending the `UiApplication` class. In `main()`, create an instance of the new class and invoke `enterEventDispatcher()` to enable the application to receive events. In the application constructor, invoke `pushScreen()` to display the custom screen for the application. The `VideoPlaybackDemoScreen` class, which is described in step 3, represents the custom screen.

```
public class VideoPlaybackDemo extends UiApplication
{
     public static void main(String[] args)
     {
          VideoPlaybackDemo app = new VideoPlaybackDemo();
          app.enterEventDispatcher();
     }

     public VideoPlaybackDemo()
     {
          pushScreen(new VideoPlaybackDemoScreen());
     }
}
```

3. Create the framework for the custom screen by extending the `MainScreen` class.

```
private class VideoPlaybackDemoScreen extends MainScreen
{
     public VideoPlaybackDemoScreen()
     {
     }
}
```

4. In the screen constructor, in a try/catch block, create an instance of the `Player` class by invoking `Manager.createPlayer(String)`, passing in the location of the video file to play.

```
try
{
    Player player = javax.microedition.media.Manager
    .createPlayer("file:///SDCard/BlackBerry/videos/soccer1.avi");
```

```
}
catch(MediaException me)
{
    Dialog.alert(me.toString());
}
catch(IOException ioe)
{
    Dialog.alert(ioe.toString());
}
```

5. To control an aspect of playback, retrieve the appropriate `Control` object. First, invoke the `Player` object's `realize()` method to access its associated `Control` object. Next , invoke `Player.getControl()` passing in the string, `VideoControl`, to retrieve the `VideoControl` object that is associated with `Player`.

```
try
{
    Player player = javax.microedition.media.Manager
    .createPlayer("file:///SDCard/BlackBerry/videos/soccer1.avi");
    player.realize();
    VideoControl videoControl = (VideoControl) player.getControl("VideoControl");
}
catch(MediaException me)
{
    Dialog.alert(me.toString());
}
catch(IOException ioe)
{
    Dialog.alert(ioe.toString());
}
```

6. Invoke `VideoControl.initDisplayMode(int mode, Object arg)` passing an `arg` parameter specifying the UI primitive that displays the video to initialize the mode that a video field uses. Cast the returned object as a `Field` object.

   Note: You can invoke `initDisplayMode()` in different ways to return a `Field` , an `Item` for use with MIDlets, or to display a video on a `Canvas` class.

```
try
{
    Player player = javax.microedition.media.Manager
    .createPlayer("file:///SDCard/BlackBerry/videos/soccer1.avi");
    player.realize();
    VideoControl videoControl = (VideoControl) player.getControl("VideoControl");
    Field videoField = (Field)videoControl.initDisplayMode(
     VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field" );

}
catch(MediaException me)
{
    Dialog.alert(me.toString());
}
catch(IOException ioe)
```

```
{
    Dialog.alert(ioe.toString());
}
```

7. Invoke `add()` to add the returned `Field` object to your `Screen` or `Manager`, as you would add any other component to your UI.

```
try
{
    Player player = javax.microedition.media.Manager
    .createPlayer("file:///SDCard/BlackBerry/videos/soccer1.avi");
    player.realize();
    VideoControl videoControl = (VideoControl) player.getControl("VideoControl");
    Field videoField = (Field)videoControl.initDisplayMode(
     VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field" );

    add(videoField);
}
catch(MediaException me)
{
    Dialog.alert(me.toString());
}
catch(IOException ioe)
{
    Dialog.alert(ioe.toString());
}
```

8. To set the volume of playback, invoke `Player.getControl()` passing in the string, `VolumeControl`, to retrieve the `VolumeControl` object that is associated with `Player`. Invoke `VolumeControl.setLevel()` to specify the volume of playback.

```
try
{
    Player player = javax.microedition.media.Manager
    .createPlayer("file:///SDCard/BlackBerry/videos/soccer1.avi");
    player.realize();
    VideoControl videoControl = (VideoControl) player.getControl("VideoControl");
    Field videoField = (Field)videoControl.initDisplayMode(
     VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field" );

    add(videoField);

    VolumeControl volume = (VolumeControl) player.getControl("VolumeControl");
    volume.setLevel(30);
}
catch(MediaException me)
{
    Dialog.alert(me.toString());
}
catch(IOException ioe)
```

```
{
    Dialog.alert(ioe.toString());
}
```

9. Invoke `Player.start()` to start playback.

```
try
{
    Player player = javax.microedition.media.Manager
    .createPlayer("file:///SDCard/BlackBerry/videos/soccer1.avi");
    player.realize();
    VideoControl videoControl = (VideoControl) player.getControl("VideoControl");
    Field videoField = (Field)videoControl.initDisplayMode(
     VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field" );

    add(videoField);

    VolumeControl volume = (VolumeControl) player.getControl("VolumeControl");
    volume.setLevel(30);

    player.start();
}
catch(MediaException me)
{
    Dialog.alert(me.toString());
}
catch(IOException ioe)
{
    Dialog.alert(ioe.toString());
}
```

Create a BlackBerry device application that takes a photograph using the camera

1. Import the required classes and interfaces.

```
import net.rim.device.api.ui.Field;
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
import javax.microedition.media.control.VideoControl;
```

2. Invoke `Manager.createPlayer(String)` by passing a parameter that is a URI that describes the image content.

```
Player cameraPlayer = Manager.createPlayer(
     "capture://video?encoding=jpeg" );
```

3. Invoke `Player.realize()` to allow a Player to get the information that it requires to get media resources.

```
cameraPlayer.realize();
```

4. Invoke `Player.getControl()` by passing a parameter that represents the `VideoControl` class. Cast the returned object as a `VideoControl` object.

```
VideoControl videoControl = (VideoControl)cameraPlayer.getControl(
     "javax.microedition.media.control.VideoControl");
```

5. Invoke `VideoControl.initDisplayMode(int mode, Object arg)` to initialize the mode that a `videoField` uses to display the video.
6. Invoke `VideoControl.getSnapshot()` to populate a byte array with the JPEG data for the photograph taken by the camera.

   ```
   int[] imageByte = videoControl.getSnapshot(null);
   ```

   Passing null to `getSnapshot()` results in a photograph capture that uses the default camera settings.

   You can adjust theparameter for `getSnapshot()` by invoking `System.getProperty()` to determine what settings are available for use.

   ```
   String encodingString =
        System.getProperty("video.snapshot.encodings");
   ```

   Invoke `stringToKeywords()` to split the properties into separate `String` array elements.

   ```
   String[] properties =
        StringUtilities.stringToKeywords(encodingString);
   ```

   There are four properties returned by `System.getProperty()` that can be individually accessed in the `String` array returned from `stringToKeywords()`. The properties are "encoding", "width", "height" and "quality".

Create a BlackBerry device application that sends audio to a Bluetooth enabled device

1. Import the required classes.

   ```
   import javax.microedition.media.Control;
   import javax.microedition.media.Manager;
   import javax.microedition.media.Player;
   import net.rim.device.api.media.control.AudioPathControl;
   ```

2. Invoke `Manager.createPlayer(String)` passing the location of an audio file.

   ```
   Player p = javax.microedition.media.Manager
       .createPlayer("http://mycompany/test.mp3");
   ```

3. Invoke `Player.realize()` to allow a Player to get media resources.

   ```
   p.realize();
   ```

4. Invoke `Player.prefetch()`.

   ```
   p.prefetch();
   ```

5. Invoke `Player.getControls()` to get the supported `AudioPathControl`.

   ```
   AudioPathControl apc = (AudioPathControl)p.getControl(
        "net.rim.device.api.media.control.AudioPathControl" );
   ```

6. Perform one of the following tasks.

Task	Steps
Send audio to a Bluetooth® device that supports voice calls.	Invoke `AudioPathControl.setAudioPath(int)` using as a parameter the `AudioPathControl.AUDIO_PATH_BLUETOOTH` attribute to send audio to the Bluetooth device. `apc.setAudioPath (AudioPathControl.AUDIO_PATH_BLUETOOTH);`
Send audio to a Bluetooth device that supports a Bluetooth enabled device such as a Bluetooth stereo headset.	Invoke `AudioPathControl.setAudioPath(int)` using as a parameter the `AudioPathControl.AUDIO_BLUETOOTH_A2DP` attribute to send audio to the Bluetooth device. `apc.setAudioPath(AudioPathControl.AUDIO_ BLUETOOTH_A2DP);`

Create a Content Handler application that plays media

This article describes development of a Content Handler application using JSR 211 to allow applications to invoke other applications, including native BlackBerry® device applications, to handle specific content types. In registering your content handler to handle video or image content types with the SEND action, for example, a menu item will appear whenever you see a video or image in the camera preview screen, in the file explorer, and so on. When clicked, this menu item will invoke your content handler with a SEND action and a link to the video or image.

1. Import the required classes.

```
import javax.microedition.content.ContentHandlerServer;
import javax.microedition.content.RequestListener;
import javax.microedition.content.Registry;
import javax.microedition.content.Invocation;
import javax.microedition.content.ActionNameMap;
```

2. Create a class that extends `UiApplication` and implements `RequestListener`.

```
public final class SendMediaDemo extends UiApplication implements
     RequestListener {
```

3. Handle application startup.

```
public static void main(String[] args) throws Exception
{
    if(args != null && args.length > 0) {
        if (args[0].equals("startup")) {
            // Register ourselves as a content handler on startup
            register();
        }
    } else {
```

```
        // Create a new instance of the application and make the currently
        // running thread the application's event dispatching thread.
        SendMediaDemo app = new SendMediaDemo();
        app.enterEventDispatcher();
    }
}
```

4. Register the application as a RequestListener.

```
public SendMediaDemo() throws ContentHandlerException {
    // Get access to the ContentHandlerServer for this application and
    // register as a listener.
    ContentHandlerServer contentHandlerServer = Registry.getServer(CLASSNAME);
    contentHandlerServer.setListener(this);
}
```

5. Create a method to handle content handler registration. This method will use `Registry.register()` to ensure that this application can be started by external menu options to send video and images to this application.

```
private static void register() throws ContentHandlerException {
    String[] types = {"image/bmp", "image/png", "image/jpeg", "video/3gpp",
     "video/mp4"};
    String[] suffixes = {".bmp", ".png", ".jpg", ".3GP", ".mp4"};
    String[] actions = {ContentHandler.ACTION_SEND};
    String[] actionNames = {"Send to demo app"};
   ActionNameMap[] actionNameMaps = {new ActionNameMap(actions,actionNames,"en")};

    // Get access to the registry
    Registry registry = Registry.getRegistry(CLASSNAME);

    // Register as a content handler
    registry.register(CLASSNAME, types, suffixes, actions, actionNameMaps, ID,
     null);
}
```

6. Implement `RequestListener.invocationRequestNotify()`. This handler will be invoked when another application passes content to this application.

```
public void invocationRequestNotify(ContentHandlerServer server) {

    Invocation invoc = server.getRequest(false);

    if(invoc != null) {
        String type = invoc.getType();

        if(type.equals("image/bmp") || type.equals("image/png") ||
     type.equals("image/jpeg")) {
            byte[] data = getData(invoc.getURL());
            displayImage(data);
        } else if(type.equals("video/3gpp") || type.equals("video/mp4")) {
            initVideo(invoc.getURL());

            if(_videoField != null) {
```

```
                displayVideo();
            }
        } else {
            System.exit(0);
        }

        server.finish(invoc, Invocation.OK);
    }
}
```

Code Sample: Create a Content Handler application that plays media

```
package com.rim.samples.device.sendmediademo;

import java.io.*;
import javax.microedition.content.*;
import javax.microedition.io.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
import javax.microedition.io.file.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

public final class SendMediaDemo extends UiApplication implements RequestListener
{
    private static final String ID = "com.rim.samples.device.sendmediademo";
    private static final String CLASSNAME =
     "com.rim.samples.device.sendmediademo.SendMediaDemo";

    private Player _player;
    private Field _videoField;
    private VideoControl _vc;

    /**
     * Entry point for application
     * @param args Command line arguments
     */
    public static void main(String[] args)
    {
        if(args != null && args.length > 0)
        {
            if (args[0].equals("startup"))
            {
                // Register ourselves as a content handler on startup
                register();
            }
        }
        else
```

```
        {
            // Create a new instance of the application and make the currently
            // running thread the application's event dispatching thread.
            SendMediaDemo app = new SendMediaDemo();
            app.enterEventDispatcher();
        }
    }

    /**
     * Registers this application as a content handler for image files
     */
    private static void register()
    {
        String[] types = {"image/bmp", "image/png", "image/jpeg", "video/3gpp",
     "video/mp4"};
        String[] suffixes = {".bmp", ".png", ".jpg", ".3GP", ".mp4"};
        String[] actions = {ContentHandler.ACTION_SEND};
        String[] actionNames = {"Send to demo app"};
        ActionNameMap[] actionNameMaps = {new
     ActionNameMap(actions,actionNames,"en")};

        // Get access to the registry
        Registry registry = Registry.getRegistry(CLASSNAME);

        try
        {
            // Register as a content handler
            registry.register(CLASSNAME, types, suffixes, actions, actionNameMaps,
     ID, null);
        }
        catch (ContentHandlerException che)
        {
            System.out.print(che.toString());
        }
        catch (ClassNotFoundException cnfe)
        {
            System.out.print(cnfe.toString());
        }
    }

    // Constructor
    public SendMediaDemo()
    {
        try
        {
            // Get access to the ContentHandlerServer for this application and
            // register as a listener.
            ContentHandlerServer contentHandlerServer =
     Registry.getServer(CLASSNAME);
            contentHandlerServer.setListener(this);
        }
        catch(ContentHandlerException che)
```

```
        {
            System.out.println(che.toString());
        }
    }

    /**
     * RequestListener implementation
     * @param server The content handler server from which to request Invocation
     objects
     */
    public void invocationRequestNotify(ContentHandlerServer server)
    {
        Invocation invoc = server.getRequest(false);
        if(invoc != null)
        {
            String type = invoc.getType();

            if(type.equals("image/bmp") || type.equals("image/png") ||
     type.equals("image/jpeg"))
            {
                byte[] data = getData(invoc.getURL());
                displayImage(data);
            }
            else if(type.equals("video/3gpp") || type.equals("video/mp4"))
            {
                initVideo(invoc.getURL());
                if(_videoField != null)
                {
                    displayVideo();
                }
            }
            else
            {
                System.exit(0);
            }

            server.finish(invoc, Invocation.OK);
        }
    }

    /**
     * Sets the size of the video
     * @param width Width of the video display
     * @param height Height of the video display
     */
    private void setVideoSize(int width, int height)
    {
        try
        {
            if (_vc != null)
            {
                _vc.setDisplaySize(width, height);
```

```
            }
        }
        catch(MediaException pe)
        {
            System.out.println(pe.toString());
        }
    }

    /**
     * Creates and initializes a video player
     * @param url The URL of the video file to play
     */
    private void initVideo(String url)
    {
        try
        {
            _player = javax.microedition.media.Manager.createPlayer(url);
            _player.realize();

            _vc = (VideoControl) _player.getControl("VideoControl");
            if (_vc != null)
            {
                _videoField = (Field) _vc.initDisplayMode
     (VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field");
                _vc.setVisible(true);
            }
        }
        catch(MediaException pe)
        {
            System.out.println(pe.toString());
        }
        catch (IOException ioe)
        {
            System.out.println(ioe.toString());
        }
    }

    /**
     * Returns a byte array containing data representing the image at the specified
     URL
     * @param url The location of the image to display
     * @return The image data
     */
    private byte[] getData(String url)
    {
        byte[] data = new byte[0];
        try
        {
            FileConnection file = (FileConnection)Connector.open(url);
            int fileSize = (int)file.fileSize();
            data = new byte[fileSize];
            InputStream inputStream = file.openInputStream();
```

```
            inputStream.read(data);
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
        return data;
    }

    /**
     * Creates a screen and displays the image
     * @param data The data representing the image to be rendered
     */
    private void displayImage(byte [] data)
    {
        // Create image field
        Bitmap image = Bitmap.createBitmapFromBytes( data, 0, -1, 5);
        BitmapField imageField = new BitmapField( image, Field.FIELD_HCENTER);


        // Create and display screen
        MainScreen screen = new
     MainScreen(net.rim.device.api.ui.Manager.NO_VERTICAL_SCROLL);
        screen.setTitle("Send Media Demo");
        screen.add(imageField);
        pushScreen(screen);
    }

    /**
     * Creates a video screen and starts the video player
     */
    private void displayVideo()
    {
        // Create and display screen
        VideoMainScreen screen = new VideoMainScreen();
        screen.setTitle("Send Media Demo");
        screen.add(_videoField);
        pushScreen(screen);

        try
        {
            // Start media player
            _player.start();
        }
        catch(MediaException pe)
        {
            System.out.println(pe.toString());
        }
    }

    /**
     * A main screen in which to play video files
```

```
     */
    final private class VideoMainScreen extends MainScreen
    {
        // Constructor
        VideoMainScreen()
        {
            super(net.rim.device.api.ui.Manager.NO_VERTICAL_SCROLL);
        }

        /**
         * @see net.rim.device.api.ui.Manager#sublayout(int,int)
         */
        protected void sublayout(int width, int height)
        {
            setVideoSize(Display.getWidth(), Display.getHeight());
            super.sublayout(width, height);
        }

        /**
         * @see net.rim.device.api.ui.Screen#onClose()
         */
        public boolean onClose()
        {
            _player.close();
            return super.onClose();
        }
    }
}
```

Specifying the features of a BlackBerry device application that plays media

Querying media playback features that a BlackBerry device application supports

Your BlackBerry® device application can query a variety of playback features that include the video display mode, the location of the video with respect to the display canvas, the x and y-coordinates of video with respect to the video display UI object, and audio levels.

Specify the parameters for video playback within a BlackBerry device application

1. Import the required classes.

```
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
import javax.microedition.media.control.VideoControl;
```

2. Initialize and prepare an instance of a `Player` object.

```
Player player =
     Manager.createPlayer("file:///SDCard/BlackBerry/videos/myvid.mp4");
player.realize();
```

3. Get the video control.

```
VideoControl videoControl = (VideoControl)
     player.getControl("VideoControl");
videoControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO, this);
```

4. To embed the video control in the canvas, specify the properties of the video control.

```
videoControl.setDisplayLocation(20, 30);
videoControl.setDisplaySize(160, 90);
videoControl.setVisible(true);
```

5. The `VideoControl` interface is implemented to give a BlackBerry® device application a variety of video playback support features. Perform one of the following actions:
   - Control the mode of video display through the use of USE_GUI_PRIMITIVE or USE_DIRECT_VIDEO.
   - Control the location of the video with respect to the canvas that displays the video.
   - Access the x-coordinate and the y-coordinate of the video with respect to the UI object that displays the video display or hides the video.

Specify the volume that a BlackBerry device application uses to play media

1. Import the required classes.

```
import net.rim.device.api.ui.Screen;
import javax.microedition.lcdui.Canvas;
import javax.microedition.media.control.VolumeControl;
```

2. Specify the volume of the media application.

```
VolumeControl volume = (VolumeControl)
     player.getControl("VolumeControl");
volume.setLevel(80);
```

3. Perform one of the following tasks.

Task	Steps
Capture volume key events in an MIDP application.	a. Create a class that extends the `Canvas` class. b. Override the `Canvas.keyPressed()` method.
Capture volume key events in a BlackBerry® device application.	a. Create a class that extends the `Screen` class. b. Override the `Screen.keyControl()`.

Task	Steps
Implement `KeyListener` to capture and handle key events in a BlackBerry® device application.	Create a class that implements `KeyListener` and implement `keyChar()`, `keyStatus()`, `keyDown()`, `keyRepeat()` and `keyUp()`. Use `keyDown()` to process volume key presses on BlackBerry devices with media keys.

Responding to user input

Receive notification when a user presses a media key on a BlackBerry device

A BlackBerry® device application can subscribe for key events using the `KeyListener` interface. The `KeyListener` receives all key events while the application is in the foreground, and receives unhandled global keys while the application is in the background.

The media keys include volume up, volume down, forward, backwards, and a combination key that handles mute, play and pause. Some BlackBerry devices have additional dedicated media keys, which include `Keypad.KEY_FORWARD` and `Keypad.KEY_BACKWARD`. The presence of these media keys can be detected using `Keypad.hasMediaKeys()`. Some BlackBerry devices also have a "mute" key while others have a "speakerphone" key. Both of these keys are assigned the same key constant, `Keypad.KEY_SPEAKERPHONE`. To determine if a key press of `KEY_SPEAKERPHONE` is a "mute" key or a "speakerphone" key, use `Keypad.hasMuteKey()`.

1. Import the required class and interface.

```
import net.rim.device.api.ui.Keypad;
import net.rim.device.api.ui.container.MainScreen;
```

2. Create a class that extends `MainScreen`, implements `KeyListener`, and registers the listener with `addKeyListener()`.

```
class KeyListenerDemo extends MainScreen implements KeyListener {
    public KeyListenerDemo() {
        addKeyListener(this);
    }
}
```

Respond when a BlackBerry device user presses a volume key

1. Import the required classes.

```
import net.rim.device.api.ui.Screen;
import net.rim.device.api.system.Characters;
```

2. Create a class that extends the `Screen` class.
3. Override `Screen.keyControl()`.

```
protected boolean keyControl(char c, int status, int time) {
```

4. Identify the volume key that the BlackBerry device user changes.

```
if (c == Characters.CONTROL_VOLUME_UP) {
} else if (c == Characters.CONTROL_VOLUME_DOWN) {
}
```

Note: In the net.rim.device.api.media.MediaKeyListener class, the keyDown(), keyRepeat(), and keyUp() methods are invoked with Keypad.KEY_VOLUME_UP and Keypad.KEY_VOLUME_DOWN, which proves finer-grain control compared to using keyChar().

Respond when the state of a Player object changes

1. Import the required classes.

```
import javax.microedition.media.Player;
import javax.microedition.media.PlayerListener;
import javax.microedition.media.MediaException;
import java.io.IOException;
```

2. Invoke `addPlayerListener()` to listen for changes to the state of the `Player` object.

```
private void doPlay() throws IOException, MediaException {
    Player p = Manager.createPlayer("file:///SDCard/BlackBerry/music/theo.mp3");
    p.addPlayerListener(this);
    p.realize();
    p.prefetch();
    p.start();
}
```

3. Override `playerUpdate()` to send a media event to the registered `PlayerListener`.

```
public void playerUpdate(Player player, final String event, Object
     eventData) {
    if (event.equals(VOLUME_CHANGED)) {
    } else if (event.equals(STARTED )) {
    } else if (event.equals(STOPPED)) {
    } else if (event.equals(DURATION_UPDATED)) {
    } else if (event.equals(END_OF_MEDIA)) {
        //Add code for actions if the end of media is reached.
        //Release resources when end of media event is received
        player.close();
    }
}
```

Streaming media in a BlackBerry device application

To stream data from a remote source to a BlackBerry® device application, you must buffer the source and control how the media application reads the data. Extend `DataSource` and create a custom implementation of `SourceStream` to send data to the media application.

RIMM streaming video file

The RIM proprietary video format (RIMM streaming file) consists of a header, a list of frames, and a footer.

The file also contains a descriptor which stores additional metadata. The location of this descriptor depends on the recording destination. If the recording destination is a file, the descriptor is located at the end of the header, before the list of frames. If the recording destination is a stream, the descriptor is located after the list of frames in the footer.

RIM proprietary video format (RIMM streaming file)

The RIM proprietary video format consists of a header, a list of frames, and a footer. When parsing this file, all values of type `int` or `short` are little-endian.

Header

Field	Value	Size
ID tag	RIMM	4 B
version	0	3 B
descriptor location	• 0 if recording to a file • 1 if recording to a stream	1 B
descriptor	descriptor values	• 75 B if recording to a file • 0 B if recording to a stream

Frames

Field	Value	Size
stream type	• 0 if this is a frame of audio • 1 if this is a frame of video	1 B
key frame	• 1 if this is a key frame • 0 otherwise	1 b

Field	Value	Size
config frame	• 1 if this is a config frame • 0 otherwise	1 b
size	frame size, in bytes	30 b
duration	length of video, in milliseconds	2 B
data	the actual frame data	*<size>* B
stream type	• 0 if this is a frame of audio • 1 if this is a frame of video	1 B
key frame	• 1 if this is a key frame • 0 otherwise	1 b
config frame	• 1 if this is a config frame • 0 otherwise	1 b
size	frame size, in bytes	30 b
duration	length of video, in milliseconds	2 B

Note: The key frame, config frame, and size fields are stored in one 32-bit `int` with the key frame and config frame fields stored in the first two bits.

Footer

Field	Value	Size
Descriptor	descriptor values	• 75 bytes if recording to a stream • 0 bytes if recording to a file

Descriptor

Field	Value	Size
audio frames	number of audio frames	4 B
video frames	number of video frames	4 B
audio key frames	number of audio key frames	4 B
video key frames	number of video key frames	4 B
audio frame rates	number of audio frame rates (number of frame rate changes + 1)	4 B

Field	Value	Size
video frame rates	number of video frame rates (number of frame rate changes + 1)	4 B
audio size	size of audio stream in bytes	4 B
video size	size of video stream in bytes	4 B
video frame rate	the initial video frame rate, in frames per second	4 B
video max frame size	size of largest video frame, in bytes	4 B
audio duration	length of audio stream, in milliseconds	4 B
video duration	length of video stream, in milliseconds	4 B
RESERVED	undefined	20 B
width	the width of the video, in pixels	2 B
height	the height of the video, in pixels	2 B
video codec	• 2 if this video codec is mpeg4 • 5 if this video codec is H.263 • 6 if this video codec is H.264	2 B
audio codec	• 0 if this audio codec is PCM • 7 if this audio codec is AMR • 0xA if this audio codec is AAC	1 B

Buffer and play streamed media

You can use the Multimedia API to create a custom class that extends `javax.microedition.media.protocol.DataSource` to customize how data is read from your application to the BlackBerry® device media player. `DataSource` provides a `SourceStream` implementation which uses `SourceStream.read()` to transfer data.

1. Import the required classes and interfaces.

```
import java.lang.Thread;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.media.protocol.DataSource;
import javax.microedition.media.protocol.SourceStream;
import javax.microedition.io.ContentConnection;
import javax.microedition.io.file.FileConnection;
import net.rim.device.api.io.SharedInputStream;
```

2. Create a custom class that extends the abstract `javax.microedition.media.protocol.DataSource` class.

```
public final class LimitedRateStreamingSource extends DataSource {
```

3. Declare instance fields that you will need for your `DataSource` implementation. The following fields were taken from the sample bufferedplaybackdemo project.

```
/** The stream connection over which media content is passed */
private ContentConnection _contentConnection;

/** An input stream shared between several readers */
private SharedInputStream _readAhead;

/** A stream to the buffered resource */
private LimitedRateSourceStream _feedToPlayer;

/** The MIME type of the remote media file */
private String _forcedContentType;

/** The thread which retrieves the remote media file */
private ConnectionThread _loaderThread;

/** The local save file into which the remote file is written */
private FileConnection _saveFile;

/** A stream for the local save file */
private OutputStream _saveStream;
```

4. Create a constructor for your custom class.

```
LimitedRateStreamingSource(String locator) {
    super(locator);
}
```

5. Implement `javax.microedition.media.protocol.DataSource.connect()`. This method is used to open a connection to the source described by the locator and initiate communication. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
public void connect() throws IOException {
```

6. Implement `javax.microedition.media.protocol.DataSource.disconnect()`. This method is used to close the connection to the source described by the locator and free resources used to maintain the connection. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
public void disconnect() {
```

7. Implement `javax.microedition.media.protocol.DataSource.getContentType()`. This method is used to retrieve a String that describes the content-type of the media that the source is providing. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
public String getContentType() {
    return _feedToPlayer.getContentDescriptor().getContentType();
}
```

8. Implement `javax.microedition.media.protocol.DataSource.getStreams()`. This method is used to retrieve a collection of streams that this source manages. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
public SourceStream[] getStreams() {
    return new SourceStream[] { _feedToPlayer };
}
```

9. Implement `javax.microedition.media.protocol.DataSource.start()`. This method is used to initiate data-transfer and must be called before data is available for reading. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
public void start() throws IOException {
    if (_saveStream != null) {
        _loaderThread = new ConnectionThread();
        _loaderThread.start();
    }
}
```

10. Implement `javax.microedition.media.protocol.DataSource.stop()`. This method is used to stop the data-transfer. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
public void stop() throws IOException {
    // Set the boolean flag to stop the thread
    _stop = true;
}
```

11. Create an class that implements `javax.microedition.media.protocol.SourceStream`. In this example, an inner class is used to provide a stream to the buffered media resource. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
private final class LimitedRateSourceStream implements SourceStream {
```

12. Implement getSeekType() to return if media is seekable or not during playback.

```
public int getSeekType() {
    return SEEKABLE_TO_START;
}
```

13. Within this inner class, implement `javax.microedition.media.protocol.SourceStream.read()`. This method is used to reads data from the input stream into an array of bytes. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
public int read(byte[] bytes, int off, int len) throws IOException {
```

14. Within this inner class, you may also want to implement `seek()`, `tell()`, and `close()`. Please refer to the the sample bufferedplaybackdemo project for additional details.
15. Close off the inner class and create another inner class that extends `java.lang.Thread`. Use this when creating a thread to download the remote file and write it to the local file. Please refer to the the sample bufferedplaybackdemo project for additional details.

```
private final class ConnectionThread extends Thread {
```

Code sample: Streaming media in a BlackBerry device application

```
/**
 * A thread which downloads the remote file and writes it to the local file
 * From bufferedplaybackdemo\LimitedRateStreamingSource.java
 */
private final class ConnectionThread extends Thread
{
    /**
     * Download the remote media file, then write it to the local file.
     */
    public void run()
    {
        try
        {
            byte[] data = new byte[READ_CHUNK];
            int len = 0;

            // Until we reach the end of the file
            while (-1 != (len = _readAhead.read(data))) {
                _totalRead += len;

                if (!_bufferingComplete && _totalRead > getStartBuffer())
     {
                    // We have enough of a buffer to begin playback
                    _bufferingComplete = true;
                    System.out.println("Initial Buffering Complete");
                    // updateLoadStatus("Buffering Complete");
                }
                if (_stop) {
                    // Stop reading
                    return;
                }
            }

            System.out.println("Downloading Complete");
            System.out.println("Total Read: " + _totalRead);

            // If the downloaded data is not the same size
            // as the remote file, something is wrong.
            if (_totalRead != _contentConnection.getLength()) {
                System.err.println("* Unable to Download entire file *");
            }

            _downloadComplete = true;
            _readAhead.setCurrentPosition(0);

            // Write downloaded data to the local file
```

```
            while (-1 != (len = _readAhead.read(data))) {
                _saveStream.write(data);
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

Code Sample: Reading data from a buffer

When the media application reads all of the data in the buffer, the BlackBerry® device application can block read requests that the media application makes until more data is buffered. Pause read operations to make sure the specified amount of audio data exists in the buffer for playback.

```
public int read(byte[] b, int off, int len) throws IOException {
    // Read from a SharedInputStream that is shared
    // with the same Stream that the ConnectionThread
    // is downloading to. Although both streams handle
    // the same data, each stream has a separate read location.
    // The ConnectionThread works to keep its read location
    // as far ahead of the current playback position as possible.
    System.out.println("Read Request for: " + len + " bytes");

    // limit bytes read to our readLimit.
    int readLength = len;
    if (readLength > getReadLimit()) {
        readLength = getReadLimit();
    }

    int available;
    boolean restart_pause = false;

    for (;;) {
        /*
         * A read action is be restricted by the amount of data
         * available for a read operation. The application needs
         * to download a specific amount of data before the
         * application can start to playback audio.
         * Enough data must be available to perform a full read
         * operation. In this sample, the application configures
         * the amount of data that is required to start the playback
         * of audio and to perform a read operation. The application
         * uses the ReadLimit value to control the amount of data
         * the second readerreads during a read operation.
         *
         * The application uses the PauseBytes setting to determine
         * when the second reader has read all the data currently
         * provided by the first variable reader. When this occurs,
         * the application pauses until there is enough data in the
```

```
         * buffer for a read operation.
         *
         * The application uses the RestartBytes setting to define
         * when there is enough data in the buffer to perform a
         * read operation.
         */
        available = _baseSharedStream.available();

        if (downloadComplete) {
            // Ignore all restrictions if downloading is complete
            System.out.println("Complete, Reading: " + len + " - Available: " +
     available);
            return _baseSharedStream.read(b, off, len);
        } else if (bufferingComplete) {
            if (restart_pause && available > getRestartBytes()) {
                //Perform a read operation as there is now enough data in the
//  buffer.
                System.out.println("Restarting - Available: " + available);
                restart_pause = false;
                return _baseSharedStream.read(b, off, readLength);
            } else if (!restart_pause && (available > getPauseBytes() ||
     available > readLength)) {
                //Determine if a previous read operation was paused
                //and if there is enough data in the buffer to perform a read
//  operation.
                if (available < getPauseBytes()) {
                    /*
                     * Perform a read operation, setting the
                     * pause flag so that future reads will pause
                     * until enough data for a read operation is
                     * loaded into the buffer
                     */
                    restart_pause = true;
                }
                System.out.println("Reading: " + readLength + " - Available: " +
     available);
                return _baseSharedStream.read(b, off, readLength);
            } else if (!restart_pause) {
                // The buffer does not contain enough data for
                // a read operation. Pause the read operation
                // until enough data is loaded into the buffer.
                restart_pause = true;
            }
        } else {
            // Pause the thread to allow the data to download
            // before performing another read operation.
            try {
                Thread.sleep(100);
            } catch (Exception e) {
                System.err.println(e.getMessage());
            }
```

```
        }
    }
}
```

Code Sample: Streaming media from a file on the BlackBerry device

The following code sample demonstrates how to create a BlackBerry® device application to download a media file and stream data from that media file. This could reduce the amount of time it takes the BlackBerry device media application to buffer and play the media data.

```
public void connect() throws IOException {
    System.out.println("Loading: " + getLocator());

    //Open a connection to the remote file
    s = (ContentConnection) Connector.open(getLocator(), Connector.READ);

    //Get the length of the remote file.
    System.out.println("Size: " + s.getLength());

    //Get the name of the remote file.
    int filenameStart = getLocator().lastIndexOf('/');
    int paramStart = getLocator().indexOf(';');

    if (paramStart < 0) {
        paramStart = getLocator().length();
    }

    String filename = getLocator().substring(filenameStart, paramStart);
    System.out.println("Filename: " + filename);

    //Open a connection to the file on the local file system.
    saveFile = (FileConnection) Connector.open
        ("file:///SDCard/blackberry/music" + filename, Connector.READ_WRITE);

    //If a file with the same name as the remote file doesn't exist
    //on the local system, download the file again.
    if (!saveFile.exists()) {
        saveFile.create();
    }

    //Configure the local file to be readable.
    saveFile.setReadable(true);

    //Create a shared input stream using the an input stream
    //from the local file.
    SharedInputStream fileStream = SharedInputStream.getSharedInputStream
        (saveFile.openInputStream());

    fileStream.setCurrentPosition(0);
```

```
    //If the file on the local file system is smaller than the remote file,
    //download the file again.
    if (saveFile.fileSize() < s.getLength()) {
        // didn't get it all before, download again
        saveFile.setWritable(true);
        saveStream = saveFile.openOutputStream();
        readAhead = SharedInputStream.getSharedInputStream(s.openInputStream());
    } else {
        downloadComplete = true;
        readAhead = fileStream;
        s.close();
    }

    if (forcedContentType != null) {
        feedToPlayer = new LimitedRateSourceStream(readAhead, forcedContentType);
    } else {
        feedToPlayer = new LimitedRateSourceStream(readAhead, s.getType());
    }
}
```

Code sample: Parsing a RIMM streaming video file

```
import java.io.*;
import java.util.*;
import javax.microedition.io.*;
import javax.microedition.io.file.*;

/**
 *
 */
public class KeyFrameOutputStream extends OutputStream {
    // output locations on the sd card
    private static final String OUT_DIR = "file:///SDCard/securitycam/";
    private static final String OUT_FILE = "output.frames";

    // some size constants
    private static final int HEADER_SIZE = 8;
    private static final int CHUNK_INFO_SIZE = 7;

    // parsing states
    private static final int STATE_HEADER          = 0;
    private static final int STATE_CHUNK_INFO      = 1;
    private static final int STATE_DATA            = 2;
    private static final int STATE_WAIT_FOR_NEXT   = 3;

    // member variables
    private int _state;
    private int _pos;
    private boolean _isVideoFrame;
    private boolean _isKeyFrame;
```

```
    private boolean _isConfigFrame;
    private boolean _startSaving;
    private boolean _saveFrame;
    private int _dataSize;
    private int _duration;

    // temp buffer ref
    private byte[] _buf;

    private FileConnection _file;
    private OutputStream _out;
    private WriteThread _writer;

    private boolean _reading;

    public KeyFrameOutputStream() {
        _state = STATE_HEADER;
        _pos = 0;
    }

    public void open() {
        _reading = true;
        try {
            // create the file connection for our frame destination
            FileConnection dir = (FileConnection)Connector.open( OUT_DIR );
            if( !dir.exists() ) {
                dir.mkdir();
            }
            dir.close();

            _file = (FileConnection)Connector.open( OUT_DIR + OUT_FILE );
            if( !_file.exists() ) {
                _file.create();
            } else {
                _file.truncate( 0L );
            }
            _out = _file.openOutputStream();
        } catch ( Exception e ) {
        }

        // start the write thread
        _writer = new WriteThread( _out );
        _writer.start();
    }

    public void startClosing() {
        // shuts down the write thread
        _reading = false;
        if( _writer != null ) _writer.stop();
    }

    public void write( int b ) throws IOException {
```

```
        if( _reading ) {
            switch( _state ) {
                case STATE_HEADER:
                    // read the video stream header
                    _pos++;
                    if( _pos == HEADER_SIZE ) {
                        _state = STATE_CHUNK_INFO;
                        _buf = new byte[CHUNK_INFO_SIZE];
                        _pos = 0;
                    }
                    break;
                case STATE_CHUNK_INFO:
                    // parse the information about the next chunk
                    _buf[_pos] = (byte)b;
                    _pos++;
                    if( _pos == CHUNK_INFO_SIZE ) {
                        // 1 indicates video frame, 0 indicates audio
                        _isVideoFrame =  (_buf[0] != 0);

                        // key frame and config frame flags are in the top two bits
//  of the data size value
                        _isKeyFrame =    ((_buf[4] & 0x80) != 0);
                        _isConfigFrame = ((_buf[4] & 0x40) != 0);
                        _dataSize = ((int)(_buf[4] & 0x3f) << 24) |
                                    ((int)(_buf[3] & 0xff) << 16) |
                                    ((int)(_buf[2] & 0xff) <<  8) |
                                    ((int)(_buf[1] & 0xff));

                        // duration is stored in the next two bytes
                        _duration = ((int)(_buf[6] & 0xff) << 8) |
                                    ((int)(_buf[5] & 0xff));

                        // we want the config frame to be the first frame in our
//  output file
                        if( !_startSaving ) {
                            if( _isVideoFrame && _isConfigFrame ) {
                                _startSaving = true;
                            }
                        }

                        // after that only save the key frames
                        _saveFrame = _startSaving && _isVideoFrame
     && ( _isConfigFrame || _isKeyFrame );

                        _state = STATE_DATA;
                        if( _saveFrame ) {
                            _buf = new byte[_dataSize];
                        }
                        _pos = 0;
                    }
                    break;
                case STATE_DATA:
```

```
                    // buffer the frame for writing to file
                    if( _saveFrame ) _buf[_pos] = (byte)b;
                    _pos++;
                    if( _pos == _dataSize ) {
                        if( _saveFrame ) {
                            _writer.addFrame( _buf );
                        }
                        _state = STATE_WAIT_FOR_NEXT;
                        _buf = new byte[CHUNK_INFO_SIZE];
                        _pos = 0;
                    }
                    break;
                case STATE_WAIT_FOR_NEXT:
                    // skip over the chunk footer
                    _pos++;
                    if( _pos == CHUNK_INFO_SIZE ) {
                        _state = STATE_CHUNK_INFO;
                        _buf = new byte[CHUNK_INFO_SIZE];
                        _pos = 0;
                    }
                    break;
            }
        }
    }

    public void close() throws IOException {
        // shut down the write thread and close our file
        try {
            _writer.join();
        } catch ( InterruptedException ie ) {
        }
        _out.close();
        _file.close();
    }

    private static final class WriteThread extends Thread {
        // writes key frames to a file as they are found by our parser
        private Vector _frames;
        private boolean _running;
        private OutputStream _out;

        public WriteThread( OutputStream out ) {
            _frames = new Vector();
            _running = true;
            _out = out;
        }

        public void run() {
            for( ;; ) {
                ByteArray frame = null;
                synchronized( this ) {
                    if( _frames.size() > 0 ) {
```

```
                        frame = (ByteArray)_frames.elementAt( 0 );
                        if( frame == null ) break;
                        _frames.removeElementAt( 0 );
                    } else {
                        if( !_running ) break;
                        try {
                            wait();
                            if( _running ) continue;
                        } catch ( InterruptedException ie ) {
                        }
                    }
                }

                if( frame == null ) break;

                try {
                    byte[] bytes = frame.array;
                    _out.write( bytes, 0, bytes.length );
                    _out.flush();
                } catch ( Exception e ) {
                }
            }
        }

        public synchronized void addFrame( byte[] frame ) {
            _frames.addElement( new ByteArray( frame ) );
            notifyAll();
        }

        public synchronized void stop() {
            _running = false;
            notifyAll();
        }
    }

    private static final class ByteArray {
        public byte[] array;
        public ByteArray( byte[] array ) {
            this.array = array;
        }
    }
}
```

Recording media by using a BlackBerry device application

Record audio in a BlackBerry device application

A BlackBerry® device can record audio in four formats: Adaptive Multi-Rate (AMR), 8 kHz mono-16-bit pulse code modulation (PCM), GSM with BlackBerry devices operating on GSM networks, and QCP with BlackBerry devices operating on CDMA networks. The default audio recording format is AMR.

1. Import the required classes.

```
import java.lang.Thread;
import javax.microedition.media.Manager;
import java.io.ByteArrayOutputStream;
import javax.microedition.media.Player;
import javax.microedition.media.control.RecordControl;
```

2. In a class that extends `Thread`, create a variable of type `Player`, a variable of type `RecordControl` for recording media from a `Player`, a variable of type `ByteArrayOutputStream` for the audio stream, and a byte array variable to store the `OutputStream` data. Note that you are not required to record audio in a separate thread because recording operations are threaded by design.

```
final class AudioRecorderThread extends Thread
{
    private Player _player;
    private RecordControl _rcontrol;
    private ByteArrayOutputStream _output;
    private byte _data[];
```

3. Create a class constructor.

```
AudioRecorderThread(){}
```

4. In a `try` block, in your implementation of the `run()` method, invoke `Manager.createPlayer(String locator)` using as a parameter a value that specifies the encoding to use to record audio. You can use the following supported locator strings.
   - AMR: `capture://audio` or `capture://audio?encoding=amr` or `capture://audio?encoding=audio/amr`
   - PCM: `capture://audio?encoding=pcm` or `capture://audio?encoding=audio/basic`
   - GSM: `capture://audio?encoding=gsm` or `capture://audio?encoding=audio/x-gsm`
   - QCP: `capture://audio?encoding=audio/qcelp`
5. Create a `Player` object by invoking `createPlayer()` to capture audio.

```
public void run() {
    try {
        _player = Manager.createPlayer("capture://audio");
```

6. Invoke `Player.realize()`.

```
_player.realize();
```

7. Invoke `Player.getControl()` to obtain the controls for recording media from a `Player`.

```
_rcontrol = (RecordControl)_player.getControl("RecordControl");
```

8. Create a `ByteArrayOutputStream` to record the audio stream. Note that you can also record directly to a file specified by a URL.

```
_output = new ByteArrayOutputStream();
```

9. Invoke `RecordControl.setRecordStream()` to set the output stream to which the BlackBerry device application records data.

```
_rcontrol.setRecordStream(_output);
```

10. Invoke `RecordStore.startRecord()` to start recording the audio and start playing the media from the `Player`.

```
_rcontrol.startRecord();
_player.start();
```

11. In a catch block, specify actions to perform if an exception occurs.

```
} catch (final Exception e){
    //Perform actions
}
```

12. Create a try block in your implementation of the stop method, and then invoke `RecordControl.commit()` to stop recording audio.

```
public void stop() {
     try {
        _rcontrol.commit()
```

13. Invoke `ByteArrayOutputStream.toByteArray()` to write the audio data from the `OutputStream` to a byte array.

```
_data = _output.toByteArray();
```

14. Invoke `ByteArrayOutputStream.close()` and `Player.close()` to close the `OutputStream` and `Player`.

```
_output.close();
_player.close();
```

15. In a catch block, specify actions to perform if an exception occurs.

```
} catch (Exception e) {
    //Perform actions
}
```

Code sample: Recording audio from a Player

```
final class AudioRecorderThread extends Thread {
    private Player _player;
    private RecordControl _rcontrol;
    private ByteArrayOutputStream _output;
    private byte _data[];
    AudioRecorderThread() {}

    private int getSize() {
        return (_output != null ? _output.size() : 0);
    }

    private byte[] getAudioBuffer() {
       return _data;
    }

    public void run() {
        try {
            // Create a Player that records live audio.
            _player = Manager.createPlayer("capture://audio");
            _player.realize();

            // Get the RecordControl, configure the record stream,
            _rcontrol = (RecordControl)_player.getControl("RecordControl");

            //Create a ByteArrayOutputStream to record the audio stream.
            _output = new ByteArrayOutputStream();
            _rcontrol.setRecordStream(_output);
            _rcontrol.startRecord();
            _player.start();
        } catch (final Exception e) {
            UiApplication.getUiApplication().invokeAndWait(new Runnable() {
                public void run() {
                    Dialog.inform(e.toString());
                }
            });
        }
    }

    public void stop() {
        try {
            //Stop recording, record data from the OutputStream,
            //close the OutputStream and player.
            _rcontrol.commit();
            _data = _output.toByteArray();
            _output.close();
            _player.close();
        } catch (Exception e) {
            synchronized (UiApplication.getEventLock()) {
```

```
            Dialog.inform(e.toString());
        }
    }
}
```

Record video using the BlackBerry video recording application

1. Import the required classes.

```
import net.rim.blackberry.api.invoke.CameraArguments;
import net.rim.blackberry.api.invoke.Invoke;
```

2. Create a `CameraArguments` object.

```
CameraArguments vidargs = new
     CameraArguments(CameraArguments.ARG_VIDEO_RECORDER);
```

3. Invoke `invokeApplication()` to start the video recording application.

```
Invoke.invokeApplication( Invoke.APP_TYPE_CAMERA, vidargs );
```

Using the Multimedia API to record video

1. Import the required classes.

```
import java.io.OutputStream;
import net.rim.device.api.ui.Field;
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
import javax.microedition.media.control.VideoControl;
import javax.microedition.media.control.RecordControl;
```

2. Extend `MainScreen` to create a video recording screen. Note that you could also use the RIM UI framework or the MIDP UI framework.

```
private static final class VideoRecordingScreen extends MainScreen {
```

3. Create a video media player to capture video. Note that `System.getProperty("video.encodings")` can be used to detect video encoding property values and create a URI that describes the capture of video content in the form of `"capture://video?" + encoding`.

```
Player _player = javax.microedition.media.Manager.createPlayer(
     "capture://video?encoding=video/3gpp" );
```

4. Start the `Player`

```
_player.start();
```

5. Obtain a `VideoControl` object to control the placement of the video viewfinder, and a `RecordControl` object to control the recording state.

```
VideoControl _vc  = (VideoControl)_player.getControl( "VideoControl"
    );
RecordControl _rc = (RecordControl)_player.getControl( "RecordControl" );
```

6. There are different ways to initialize the video display mode. The following technique shows how to handle the process if you want a `Field` that represents the video viewfinder region for use in your BlackBerry® device UI.

```
Field videoField = (Field)_vc.initDisplayMode(
     VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field" );
add( videoField );
```

7. To record to a video stream.
   a. Open a `FileConnection` to the streaming video file.

```
FileConnection _fc = (FileConnection)Connector.open(
     "file:///SDCard/BlackBerry/videos/mmapi_stream.sbv" );
```

   b. Create the streaming file if it does not exist.

```
if( !_fc.exists() ) {
    _fc.create();
}
```

   c. Erase the streaming file.

```
_fc.truncate( 0 );
```

   d. Open the `OutputStream`, associate the recording stream with the `OutputStream`, and start recording the stream. When recording to an OutputStream, the data will be written out in a proprietary format which makes it easy to parse audio and video frames.

```
OutputStream _out = _fc.openOutputStream();
_rc.setRecordStream( _out );
 _rc.startRecord();
```

   e. After the video stream has been captured, stop and commit the recording, and close the streams.

```
_rc.stopRecord();
_rc.commit();
_out.close();
_fc.close();
```

8. To record a video to a file.
   a. Set the output location where the data will be recorded.

```
RecordControl _rc.setRecordLocation(
     "file:///SDCard/BlackBerry/videos/mmapi_video.3gp" );
```

   b. Start recording the video in 3GP format.

```
_rc.startRecord();
```

c. After the video has been captured, stop and commit the recording.

```
_rc.stopRecord();
_rc.commit();
```

Code Sample: Using the Multimedia API to record video

```
import java.io.*;

import javax.microedition.media.*;
import javax.microedition.media.control.*;
import javax.microedition.io.*;
import javax.microedition.io.file.*;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.util.*;
import net.rim.device.api.system.*;
import net.rim.device.api.media.*;

/*
 * the entry class
 */
public class VideoRecordingApplication extends UiApplication
{
    /*
     * working constants hard coded to create video files on the SD media card
     */
    private static final String VIDEO_FILE =
     "file:///SDCard/BlackBerry/videos/mmapi_rimlet.3GP";
    private static final String STREAM_VIDEO_FILE =
     "file:///SDCard/BlackBerry/videos/mmapi_stream.sbv";

    /*
     * video recording screen object used in several places within
     */
    private static VideoRecordingScreen _vrs;

    /*
     * constructor
    */
    private VideoRecordingApplication()
    {
        /*
         * to select the video encoding
         */
        pushScreen( new ChooseEncodingScreen() );
```

```
    }

    /*
     * main() entry point
    */
    public static void main( String[] args )
    {
        /*
         * fire up the event dispatcher loop
         */
        new VideoRecordingApplication().enterEventDispatcher();
    }

    /*
     * select video encoding screen
     * called by the VideoRecordingApplication class constructor
     */
    private static final class ChooseEncodingScreen extends MainScreen
    {
        private static ObjectChoiceField _encodings;

        /*
         * constructor
         */
        public ChooseEncodingScreen()
        {
            super();
            setTitle( "Choose an encoding" );

            /*
             * load String[] array with all MMAPI video encoding system property
     values
             */
            String[] encodings = getEncodings();

            /*
             * prep for display
            */
            _encodings = new ObjectChoiceField( "Encoding:", encodings, 0 );

            /*
             * add field to this screen's manager
            */
            add( _encodings );

            /*
             * create a menu item to start recording using the selected encoding
     scheme
            */
            addMenuItem( new MenuItem( "Go", 0, 0 )
            {
                public void run()
```

```
            {
                /*
                 * create and display the video recording screen, passing the
selected video encoding scheme
                */
                _vrs = new VideoRecordingScreen( (String)_encodings.getChoice(
_encodings.getSelectedIndex() ) );
                UiApplication.getUiApplication().pushScreen( _vrs );
            }
        } );
    }

    /*
     * returns all MMAPI video encoding system property values
     */
    private String[] getEncodings()
    {
        String[] encodings = new String[0];

        /*
         * load String with all video.encodings property values
         * each value is separated by a space
         */
        String encodingsString = System.getProperty( "video.encodings" );

        /*
         * determine where the first value ends
         */
        int space = encodingsString.indexOf( ' ' );

        /*
         * loop through values, end when a new field separator is not found
         */
        while( space != -1 )
        {
            /*
             * add a new String array element that contains the system property
value
             */
            Arrays.add( encodings, encodingsString.substring( 0, space ) );

            /*
             * remove the last property value from the encoding string
             */
            encodingsString = encodingsString.substring( space + 1 );

            /*
             * determine where the first value ends
             */
            space = encodingsString.indexOf( ' ' );
        }
```

```
            /*
             * add a new String array element that contains the final system
     property value
             */
            Arrays.add( encodings, encodingsString );

            /*
             * return the resulting String[] array
             */
            return encodings;
        }
    }

    /*
     * video recording screen displayed after the user selects "Go" from the menu
     */
    private static final class VideoRecordingScreen extends MainScreen
    {
        private static Player _player;
        private static VideoControl _vc;
        private static RecordControl _rc;
        private static boolean _visible = true;
        private static boolean _locationSet = false;
        private static RichTextField _status;
        private static CheckboxField _stream;
        private static OutputStream _out;
        private static FileConnection _fc;
        private static RichTextField _flashIndicator;

        /*
         * constructor passed the chosen video encoding property value
         */
        public VideoRecordingScreen( String encoding )
        {
            super();

            try {
                /*
                 * create a video media player to capture video using the passed
     encoding value
                 */
                _player = javax.microedition.media.Manager.createPlayer(
     "capture://video?" + encoding );

                /*
                 * try to start the player
                 */
                _player.start();

                _vc = (VideoControl)_player.getControl( "VideoControl" );
```

```
                _rc = (RecordControl)_player.getControl( "RecordControl" );

                /*
                 * Initialize the mode on how the video is displayed.
                 * This method must be called before the video can be displayed.
                 *
                 * USE_GUI_PRIMITIVE defines a mode on how the GUI is displayed.
                 * It is used in conjunction with initDisplayMode(int,
    java.lang.Object).
                 */
                Field videoField = (Field)_vc.initDisplayMode(
    VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field" );

                /*
                 * add field to this screen's manager
                 */
                add( videoField );

                _status = new RichTextField( "Idle" );
                add( _status );

                _stream = new CheckboxField( "Use OutputStream", false );
                add( _stream );

            } catch ( Exception e ) {
                System.out.println( "Exception in VideoScreen constructor" );
            }

            addMenuItem( new MenuItem( "Start Record", 0, 0 )
            {
                public void run()
                {
                    startRecord();
                }
            } );

            addMenuItem( new MenuItem( "Stop Record", 0, 0 )
            {
                public void run()
                {
                    stopRecord();
                }
            } );

            addMenuItem( new MenuItem( "Commit Record", 0, 0 )
            {
                public void run()
                {
                    commit();
                }
            } );
```

```
            addMenuItem( new MenuItem( "Toggle Visibility", 0, 0 )
            {
                public void run()
                {
                    toggleVisibility();
                }
            } );

            addMenuItem( new MenuItem( "Play Recording", 0, 0 )
            {
                public void run()
                {
                    //InputStream recording = _stream.getChecked() ? new
//  ByteArrayInputStream( _out.toByteArray() ) : null;
                    Application.getApplication().invokeLater( new ShowVideoRunnable(
     null ) );
                }
            } );

            /*
             * this loads the streamed video into memory then does nothing with it
             */
            addMenuItem( new MenuItem( "Analyze Recording Stream", 0, 0 )
            {
                public void run()
                {
                    try {
                        /*
                         * get a handle to the streaming video file located on the
     SDCard (declared near the start of this file)
                         */
                        _fc = (FileConnection)Connector.open( STREAM_VIDEO_FILE );

                        /*
                         * determine the file size
                         */
                        long size = _fc.fileSize();

                        /*
                         * declare a block of ram to store the file
                         */
                        byte[] file = new byte[(int)size];

                        /*
                         * open the stream for read
                         */
                        InputStream in = _fc.openInputStream();

                        /*
                         * load the file into memory
                         */
                        in.read( file );
```

```
                    } catch ( Exception e ) {
                    }
                }
            } );
        }

        /*
         * decipher and act upon shortcut keys
         */
        protected boolean keyChar( char c, int status, int time )
        {
            switch( c ) {
                case 's':
                    startRecord();
                    return true;
                case 'x':
                    stopRecord();
                    return true;
                case 'c':
                    commit();
                    return true;
                case 'v':
                    toggleVisibility();
                    return true;
                case 'r':
                    reset();
                    return true;
                default:
                    return false;
            }
        }

        /*
         * record captured video
         */
        private void startRecord()
        {
            try {
                /*
                 * _locationSet == false by default
                 */
                if( !_locationSet )
                {
                    /*
                     * test if "Use OutputStream" CheckboxField is set
                     */
                    if( _stream.getChecked() )
                    {
                        try {
                            _fc = (FileConnection)Connector.open( STREAM_VIDEO_FILE
    );
```

```
                        /*
                         * create streaming file if it does not exist
                         */
                        if( !_fc.exists() )
                        {
                            _fc.create();
                        }

                        /*
                         * zap the file
                         */
                        _fc.truncate( 0 );

                        /*
                         * ready to write video stream
                         */
                        _out = _fc.openOutputStream();
                    } catch ( Exception e ) {
                        return;
                    }

                    _rc.setRecordStream( _out );

                } else {
                    /*
                     * FileConnection: Set the output stream where the data will
be recorded.
                     * The locator specifying where the recorded media will be
saved.
                     * The locator must be specified as a URL.
                     */
                    _rc.setRecordLocation( VIDEO_FILE );
                }
                _locationSet = true;
            }

            /*
             * Start recording the media.
             */
            _rc.startRecord();
            _status.setText( "Recording" );
        } catch ( Exception e ) {
        }
    }

    private void stopRecord()
    {
        try {
            /*
             * Stop recording the media.
             * stopRecord will not automatically stop the Player.
             * It only stops the recording, putting the Player in "standby" mode
```

```
                 */
                _rc.stopRecord();
                _status.setText( "Recording stopped" );
            } catch ( Exception e ) {
            }
        }

        private void commit()
        {
            try {
                /*
                 * Complete the current recording.
                 * If the recording is in progress, commit will implicitly call
    stopRecord.
                 */
                _rc.commit();

                Dialog.alert( "Committed" );
                _locationSet = false;
                _status.setText( "Committed" );

                /*
                 * video stream was recorded
                 */
                if( _stream.getChecked() )
                {
                    /*
                     * close the stream
                     */
                    try {
                        _out.close();
                        _fc.close();
                    } catch ( Exception e ) {
                    }
                }
            } catch ( Exception e ) {
            }
        }

        /*
         * toggle video visibility
         */
        private void toggleVisibility()
        {
            try {
                _visible = !_visible;

                /*
                 * show or hide the video
                 */
                _vc.setVisible( _visible );
            } catch ( Exception e ) {
```

```
            }
        }

        /*
         * Erase the current recording
         */
        private void reset()
        {
            try {
                _rc.reset();
                _status.setText( "Reset called, idle" );
            } catch ( Exception e ) {
            }
        }

        /*
         * hide the video
         */
        public void hide()
        {
            try {
                _visible = false;
                _vc.setVisible( _visible );
            } catch ( Exception e ) {
            }
        }

        /*
         * show the video
         */
        public void show()
        {
            try {
                _visible = true;
                _vc.setVisible( _visible );
            } catch ( Exception e ) {
            }
        }
    }

    /*
     * ShowVideoRunnable() called within VideoRecordingScreen() through the "Play
     Recording" menu
     */
    private static final class VideoPlaybackScreen extends MainScreen
    {
        private Player _video;
        private VideoControl _vc;

        public VideoPlaybackScreen( InputStream vid )
        {
            super();
```

```
        try {
            /*
             * existing video stream passed
             */
            if( vid != null )
            {
                /*
                 * Create a Player to play back media from an InputStream
(streamed video)
                 */
                _video = javax.microedition.media.Manager.createPlayer( vid,
"video/x-rim" );
            } else {
                /*
                 * Create a Player from an input locator (a standard video file
in .3GP format)
                 */
                _video = javax.microedition.media.Manager.createPlayer(
VIDEO_FILE );
            }

            /*
             * Constructs portions of the Player without acquiring the scarce
and exclusive resources.
             */
            _video.realize();

            /*
             * Obtain the object that implements the specified Control
interface.
             */
            _vc = (VideoControl)_video.getControl( "VideoControl" );

            /*
             * Initialize the mode on how the video is displayed
             */
            Field vField = (Field)_vc.initDisplayMode(
VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field" );
            add( vField );

            VolumeControl vol = (VolumeControl)_video.getControl(
"VolumeControl" );
            vol.setLevel( 30 );

        } catch ( Exception e ) {
            System.out.println( "Exception while showing video" );
        }
    }

    protected void onVisibilityChange( boolean visible )
    {
```

```
            if( visible ) {
                try {
                    /*
                     * Starts the Player as soon as possible.
                     */
                    _video.start();
                } catch ( Exception e ) {
                }
            }
        }

        public boolean onClose() {
            try {
                /*
                 * Close the Player and release its resources.
                 */
                _video.close();
            } catch ( Exception e ) {
            }
            _vrs.show();
            return super.onClose();
        }
    }

    /*
     * called within VideoRecordingScreen() through the "Play Recording" menu
     */
    private static final class ShowVideoRunnable implements Runnable
    {
        private InputStream _in;

        public ShowVideoRunnable( InputStream in )
        {
            _in = in;
        }

        public void run()
        {
            /*
             * hide the VideoRecordingScreen
             */
            _vrs.hide();

            /*
             * Retrieves this UI application object.
             */
            UiApplication ui = UiApplication.getUiApplication();

            /*
             * handle video playback
             */
            ui.pushScreen( new VideoPlaybackScreen( _in ) );
```

```
        }
    }
}
```

Playing media in the BlackBerry device media application

On the BlackBerry® devices that include the BlackBerry device media application, a BlackBerry device application can use the `javax.microedition.content` and `net.rim.device.api.content` packages to start the BlackBerry device media application with or without passing media content to load.

Start the media application with or without content

1. Import the required class.

```
import javax.microedition.content;
```

2. Invoke `Registry.getRegistry(classname)` where the classname parameter is the name of the class in the BlackBerry® device application that extends `javax.microedition.midlet.MIDlet`, `net.rim.device.api.system.Application`, or `net.rim.device.api.ui.UiApplication`. Store a reference to the returned object in a `Registry` object.

```
Registry reg = Registry.getRegistry(String classname);
```

3. Perform one of the following tasks.

Task	Steps
Start the media application with no content.	Create a new instance of an `Invocation` object. Store a reference to the object in an `Invocation` object. `Invocation invocation = new Invocation(null, null, BlackBerryContentHandler.ID_MEDIA_CONTENT_HANDLER)`
Start the media application with content	Create a new instance of an `Invocation` object. Use a media type supported by the media application as a parameter, and store a reference to the object in an `Invocation` object. `Invocation invocation = new Invocation("file://...")`

4. To start the media application, invoke `Registry.invoke(Invocation invocation)` by passing an `Invocation` object.

```
reg.invoke(invocation)
```

Play audio in the BlackBerry Browser

1. Import the required classes.

```
import net.rim.blackberry.api.browser.Browser;
import net.rim.blackberry.api.browser.BrowserSession;
```

2. Invoke `Browser.getDefaultSession()`.

```
BrowserSession soundclip = Browser.getDefaultSession();
```

3. Invoke `BrowserSession.displaypage()`.

```
soundclip.displayPage("file:///SDCard/BlackBerry/music/yourFile
    .mp3");
```

Play a video in the BlackBerry Browser

1. Import the required classes.

```
import net.rim.blackberry.api.browser.Browser;
import net.rim.blackberry.api.browser.BrowserSession;
```

2. Invoke `Browser.getDefaultSession()`.

```
BrowserSession videoclip = Browser.getDefaultSession();
```

3. Invoke `BrowserSession.displaypage()`.

```
videoclip.displayPage("file:///SDCard/BlackBerry/Video/soccergame
    .avi");
```

Working with images from the camera application 2

Start the camera from a BlackBerry device application

You can create a BlackBerry® device application that uses the `Invoke` class to start the camera application so a BlackBerry device user can take a photograph.

1. Import the required class.

```
import net.rim.blackberry.api.invoke.Invoke;
```

2. Invoke `invokeApplication()` to invoke the camera application.

```
Invoke.invokeApplication(Invoke.APP_TYPE_CAMERA, new
     CameraArguments());
```

Create a Content Handler application that plays media

This article describes development of a Content Handler application using JSR 211 to allow applications to invoke other applications, including native BlackBerry® device applications, to handle specific content types. In registering your content handler to handle video or image content types with the SEND action, for example, a menu item will appear whenever you see a video or image in the camera preview screen, in the file explorer, and so on. When clicked, this menu item will invoke your content handler with a SEND action and a link to the video or image.

1. Import the required classes.

```
import javax.microedition.content.ContentHandlerServer;
import javax.microedition.content.RequestListener;
import javax.microedition.content.Registry;
import javax.microedition.content.Invocation;
import javax.microedition.content.ActionNameMap;
```

2. Create a class that extends `UiApplication` and implements `RequestListener`.

```
public final class SendMediaDemo extends UiApplication implements
     RequestListener {
```

3. Handle application startup.

```
public static void main(String[] args) throws Exception
{
    if(args != null && args.length > 0) {
        if (args[0].equals("startup")) {
            // Register ourselves as a content handler on startup
            register();
        }
    } else {
        // Create a new instance of the application and make the currently
        // running thread the application's event dispatching thread.
```

```
        SendMediaDemo app = new SendMediaDemo();
        app.enterEventDispatcher();
    }
}
```

4. Register the application as a RequestListener.

```
public SendMediaDemo() throws ContentHandlerException {
    // Get access to the ContentHandlerServer for this application and
    // register as a listener.
    ContentHandlerServer contentHandlerServer = Registry.getServer(CLASSNAME);
    contentHandlerServer.setListener(this);
}
```

5. Create a method to handle content handler registration. This method will use `Registry.register()` to ensure that this application can be started by external menu options to send video and images to this application.

```
private static void register() throws ContentHandlerException {
    String[] types = {"image/bmp", "image/png", "image/jpeg", "video/3gpp",
     "video/mp4"};
    String[] suffixes = {".bmp", ".png", ".jpg", ".3GP", ".mp4"};
    String[] actions = {ContentHandler.ACTION_SEND};
    String[] actionNames = {"Send to demo app"};
   ActionNameMap[] actionNameMaps = {new ActionNameMap(actions,actionNames,"en")};

    // Get access to the registry
    Registry registry = Registry.getRegistry(CLASSNAME);

    // Register as a content handler
    registry.register(CLASSNAME, types, suffixes, actions, actionNameMaps, ID,
     null);
}
```

6. Implement `RequestListener.invocationRequestNotify()`. This handler will be invoked when another application passes content to this application.

```
public void invocationRequestNotify(ContentHandlerServer server) {

    Invocation invoc = server.getRequest(false);

    if(invoc != null) {
        String type = invoc.getType();

        if(type.equals("image/bmp") || type.equals("image/png") ||
     type.equals("image/jpeg")) {
            byte[] data = getData(invoc.getURL());
            displayImage(data);
        } else if(type.equals("video/3gpp") || type.equals("video/mp4")) {
            initVideo(invoc.getURL());

            if(_videoField != null) {
                displayVideo();
            }
```

```
        } else {
            System.exit(0);
        }

        server.finish(invoc, Invocation.OK);
    }
}
```

Code Sample: Create a Content Handler application that plays media

```
package com.rim.samples.device.sendmediademo;

import java.io.*;
import javax.microedition.content.*;
import javax.microedition.io.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
import javax.microedition.io.file.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

public final class SendMediaDemo extends UiApplication implements RequestListener
{
    private static final String ID = "com.rim.samples.device.sendmediademo";
    private static final String CLASSNAME =
     "com.rim.samples.device.sendmediademo.SendMediaDemo";

    private Player _player;
    private Field _videoField;
    private VideoControl _vc;

    /**
     * Entry point for application
     * @param args Command line arguments
     */
    public static void main(String[] args)
    {
        if(args != null && args.length > 0)
        {
            if (args[0].equals("startup"))
            {
                // Register ourselves as a content handler on startup
                register();
            }
        }
        else
        {
            // Create a new instance of the application and make the currently
```

```
            // running thread the application's event dispatching thread.
            SendMediaDemo app = new SendMediaDemo();
            app.enterEventDispatcher();
        }
    }

    /**
     * Registers this application as a content handler for image files
     */
    private static void register()
    {
        String[] types = {"image/bmp", "image/png", "image/jpeg", "video/3gpp",
     "video/mp4"};
        String[] suffixes = {".bmp", ".png", ".jpg", ".3GP", ".mp4"};
        String[] actions = {ContentHandler.ACTION_SEND};
        String[] actionNames = {"Send to demo app"};
        ActionNameMap[] actionNameMaps = {new
     ActionNameMap(actions,actionNames,"en")};

        // Get access to the registry
        Registry registry = Registry.getRegistry(CLASSNAME);

        try
        {
            // Register as a content handler
            registry.register(CLASSNAME, types, suffixes, actions, actionNameMaps,
     ID, null);
        }
        catch (ContentHandlerException che)
        {
            System.out.print(che.toString());
        }
        catch (ClassNotFoundException cnfe)
        {
            System.out.print(cnfe.toString());
        }
    }

    // Constructor
    public SendMediaDemo()
    {
        try
        {
            // Get access to the ContentHandlerServer for this application and
            // register as a listener.
            ContentHandlerServer contentHandlerServer =
     Registry.getServer(CLASSNAME);
            contentHandlerServer.setListener(this);
        }
        catch(ContentHandlerException che)
        {
            System.out.println(che.toString());
```

```
        }
    }

    /**
     * RequestListener implementation
     * @param server The content handler server from which to request Invocation
     objects
     */
    public void invocationRequestNotify(ContentHandlerServer server)
    {
        Invocation invoc = server.getRequest(false);
        if(invoc != null)
        {
            String type = invoc.getType();

            if(type.equals("image/bmp") || type.equals("image/png") ||
     type.equals("image/jpeg"))
            {
                byte[] data = getData(invoc.getURL());
                displayImage(data);
            }
            else if(type.equals("video/3gpp") || type.equals("video/mp4"))
            {
                initVideo(invoc.getURL());
                if(_videoField != null)
                {
                    displayVideo();
                }
            }
            else
            {
                System.exit(0);
            }

            server.finish(invoc, Invocation.OK);
        }
    }

    /**
     * Sets the size of the video
     * @param width Width of the video display
     * @param height Height of the video display
     */
    private void setVideoSize(int width, int height)
    {
        try
        {
            if (_vc != null)
            {
                _vc.setDisplaySize(width, height);
            }
        }
```

```
        catch(MediaException pe)
        {
            System.out.println(pe.toString());
        }
    }

    /**
     * Creates and initializes a video player
     * @param url The URL of the video file to play
     */
    private void initVideo(String url)
    {
        try
        {
            _player = javax.microedition.media.Manager.createPlayer(url);
            _player.realize();

            _vc = (VideoControl) _player.getControl("VideoControl");
            if (_vc != null)
            {
                _videoField = (Field) _vc.initDisplayMode
     (VideoControl.USE_GUI_PRIMITIVE, "net.rim.device.api.ui.Field");
                _vc.setVisible(true);
            }
        }
        catch(MediaException pe)
        {
            System.out.println(pe.toString());
        }
        catch (IOException ioe)
        {
            System.out.println(ioe.toString());
        }
    }

    /**
     * Returns a byte array containing data representing the image at the specified
     URL
     * @param url The location of the image to display
     * @return The image data
     */
    private byte[] getData(String url)
    {
        byte[] data = new byte[0];
        try
        {
            FileConnection file = (FileConnection)Connector.open(url);
            int fileSize = (int)file.fileSize();
            data = new byte[fileSize];
            InputStream inputStream = file.openInputStream();
            inputStream.read(data);
        }
```

```
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
        return data;
    }

    /**
     * Creates a screen and displays the image
     * @param data The data representing the image to be rendered
     */
    private void displayImage(byte [] data)
    {
        // Create image field
        Bitmap image = Bitmap.createBitmapFromBytes( data, 0, -1, 5);
        BitmapField imageField = new BitmapField( image, Field.FIELD_HCENTER);


        // Create and display screen
        MainScreen screen = new
     MainScreen(net.rim.device.api.ui.Manager.NO_VERTICAL_SCROLL);
        screen.setTitle("Send Media Demo");
        screen.add(imageField);
        pushScreen(screen);
    }

    /**
     * Creates a video screen and starts the video player
     */
    private void displayVideo()
    {
        // Create and display screen
        VideoMainScreen screen = new VideoMainScreen();
        screen.setTitle("Send Media Demo");
        screen.add(_videoField);
        pushScreen(screen);

        try
        {
            // Start media player
            _player.start();
        }
        catch(MediaException pe)
        {
            System.out.println(pe.toString());
        }
    }

    /**
     * A main screen in which to play video files
     */
    final private class VideoMainScreen extends MainScreen
```

```
    {
        // Constructor
        VideoMainScreen()
        {
            super(net.rim.device.api.ui.Manager.NO_VERTICAL_SCROLL);
        }

        /**
         * @see net.rim.device.api.ui.Manager#sublayout(int,int)
         */
        protected void sublayout(int width, int height)
        {
            setVideoSize(Display.getWidth(), Display.getHeight());
            super.sublayout(width, height);
        }

        /**
         * @see net.rim.device.api.ui.Screen#onClose()
         */
        public boolean onClose()
        {
            _player.close();
            return super.onClose();
        }
    }
}
```

Receiving notification of file system events

You can create a BlackBerry® device application that uses the following classes and interface to detect when an image is added to or removed from a file system for a BlackBerry device. The `FileSystemJournal` class acts as a journal that tracks changes to file systems. A BlackBerry device application can query this journal to obtain information about file system events.

- `net.rim.device.api.io.file.FileSystemJournal`
- `net.rim.device.api.io.file.FileSystemJournalEntry`
- `net.rim.device.api.io.file.FileSystemJournalListener`

Detect when an image is added or removed in the BlackBerry device file system

1. Import the required classes.

```
import net.rim.device.api.system.Application;
import net.rim.device.api.io.file.FileSystemJournalListener;
```

2. Implement the `FileSystemJournalListener` interface.

```
class FileExplorerDemoJournalListener implements
     FileSystemJournalListener {
```

3. Create an instance variable that represents the last update sequence number.

```
private long _lastUSN;
```

4. Implement `FileSystemJournalListener.fileJournalChanged()` to receive notification when a new file system event occurs.

```
public void fileJournalChanged() {
```

5. Within your implementation of file`JournalChanged()`, obtain the next USN that the journal will use.

```
long nextUSN = FileSystemJournal.getNextUSN();
```

6. Create a for loop that iterates backwards from the next USN that the journal will use to the last USN that the journal will use. A best practice is to stop the iteration once the BlackBerry® device application retrieves a journal entry.

```
for (long lookUSN = nextUSN - 1; lookUSN >= _lastUSN &&
     msg == null; --lookUSN) {
```

7. In the for-loop, retrieve the journal entry that corresponds to the current USN.

```
FileSystemJournalEntry entry = FileSystemJournal.getEntry(lookUSN);
```

8. Create an IF clause that checks if the journal entry is null. If it is, a new file system journal entry does not exist.

```
if (entry == null) {
    break;
}
```

9. If a journal entry does exist, obtain the path associated with the entry.

```
String path = entry.getPath();
```

10. Create an IF clause that checks if the path for the entry is not null.

```
if (path != null) {
```

11. Create an IF clause that checks if the file system event involves an image file.

```
if (path.endsWith("png") || path.endsWith("jpg") ||
     path.endsWith("bmp") || path.endsWith("gif") ){
```

12. Perform actions if a file was added or deleted from the file system.

```
switch (entry.getEvent()) {
    case FileSystemJournalEntry.FILE_ADDED:
       //either a picture was taken or a picture was added to the BlackBerry device
//
        break;
    case FileSystemJournalEntry.FILE_DELETED:
        //a picture was removed from the BlackBerry device;
        break;
```

13. Invoke `Application.addFileSystemJournalListener()` to add the listener to the BlackBerry device application. In this example, `FileExplorerDemo` extends `UIApplication` which extends `Application`.

```
this.addFileSystemJournalListener(new
     FileExplorerDemoJournalListener());
```

Retrieve an image

1. Import the required class and interface.

```
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
```

2. Implement the following interface.
   - javax.microedition.io.file.FileConnection
3. Invoke `Connector.open()` using as a parameter the location of the image on the BlackBerry® device or the microSD card.

```
FileConnection fconn =
     (FileConnection)Connector.open("file:///store/home/user/pictures/
newfile.jpg");
FileConnection fconn = (FileConnection)Connector
    .open("file:///SDCard/pictures/newfile.jpg");
```

Move an image within the same directory of the file system

Images can be moved by renaming a selected file or directory to a new name in the same directory providing the new name is not already in use. The file or directory is renamed immediately on the actual file system upon invocation of this method. No file or directory by the original name exists after this method call.

1. Import the required class and interface.

```
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
```

2. Invoke `Connector.open()` with a parameter of the location of the image on a microSD card.

```
FileConnection fconn =
     (FileConnection)Connector.open("file:///SDCard/pictures/newfile.jpg");
```

3. Invoke `FileConnection.rename()`.

```
fconn.rename("newfilename.jpg");
```

Move an image to a different directory of the file system

1. Import the required classes and interface.

```
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
```

2. Store the name and location of the old file and the new file.

```
String oldFile = "file:///store/home/user/pictures/newfile.jpg";
String newFile = "file:///SDCard/pictures/newfile.jpg";
```

3. Create a file connection for the old and the new location of the file.

```
FileConnection source = (FileConnection) Connector.open(oldFile);
FileConnection dest = (FileConnection) Connector.open(newFile);
```

4. If the file does not exist in the destination directory, create it.

```
if (!dest.exists()) {
    dest.create();
}
```

5. Make the file in the destination directory writable.

```
dest.setWritable(true);
```

6. Open an output stream to the destination location of the file.

```
OutputStream outStream = dest.openOutputStream();
```

7. Open an input stream to the source location of the file.

```
InputStream inStream = source.openInputStream();
```

8. Create a `Byte` array that represents the size of the buffer for the `FileInputStream`.

```
byte[] Buffer = new byte[1024];
```

9. Create a `While` loop that reads data from the `InputStream` and into the `OutputStream`.

```
int length = -1;
while ((length = inStream.read(buffer)) > 0) {
    outStream.write(buffer, 0 , length);
}
```

10. Delete the source file, and then close the input stream and the output stream connections.

```
inStream.delete();
inStream.close();
outStream.close();
```

Delete an image from the file system

1. Import the required class and interface.

```
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
```

2. Invoke `Connector.open()` using as a parameter the location of the image on the BlackBerry® device or the microSD card.

```
FileConnection fconn =
     (FileConnection)Connector.open("file:///store/home/user/pictures/
newfile.jpg");
```

3. Invoke FileConnection.delete().

```
fconn.delete();
```

4. Since invoking `FileConnection.delete()` automatically flushes and closes all open input and output streams, invoke `FileConnection.close()` to prevent exceptions from occurring.

```
fconn.close();
```

Drawing and positioning images

3

Your BlackBerry® device application can use `Graphics` objects to perform drawing functions and rendering operations. Use the `Graphics` class to draw over the entire screen or on a `Field`. If your application does not contain any fields, override the `javax.microedition.lcdui.Canvas.paint(Graphics)` method to perform the painting using the specified Graphics object.

To draw in a specific `Bitmap`, a BlackBerry device application obtains a graphics context by passing the `Bitmap` object into the `Graphics` constructor. To draw over a `Bitmap` without changing the content of the `Bitmap`, create a `BitmapField` and pass the `Bitmap` object into the `BitmapField` constructor. When drawing in a field, the field's manager passes a graphics context to the fields when the fields repaint. To perform custom drawing in a field, override the `paint(Graphics g)` method when you extend the `Field` class. The `Graphics` class enables you to draw shapes, such as arcs, lines, rectangles, and circles.

Position an image

1. Import the required classes.

```
import net.rim.device.api.system.Bitmap;
import net.rim.device.api.ui.component.BitmapField;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.container.MainScreen;
```

2. Perform one of the following tasks:

Task	Steps

Use a field to position an image	a. Invoke the `Graphics()` constructor.
	`Bitmap surface = new Bitmap(100, 100);`
	`BitmapField surfaceField = new BitmapField(surface);`
	`Graphics graphics = new Graphics(surface);`
	b. Add the `BitmapField` to the screen.
	`mainScreen.add(surfaceField);`

Use the screen of an application to position an image	a. Invoke the `Screen.getGraphics()` method.
	`Graphics graphics = Screen.getGraphics();`
	b. Specify that the methods perform the drawing functions within the boundaries of the screen.
	`graphics.fillRect(10, 10, 30, 30);`
	`graphics.drawRect(15, 15, 30, 30);`

Verify that the BlackBerry device supports a color screen

1. Import the required class.

```
import net.rim.device.api.system.Display;
```

2. Invoke `Display.isColor()`.

```
if( Display.isColor() ) {
    RED    = 0xff0000;
    GREEN  = 0x00ff00;
    BLUE   = 0x5555ff;
} else {
    RED    = 0xffffff;
    GREEN  = 0xffffff;
    BLUE   = 0xffffff;
}
```

Retrieve the number of colors that the BlackBerry device screen supports

1. Import the required class.

```
import net.rim.device.api.system.Display;
```

2. Invoke `Display.getNumColors()`.

```
switch( Display.getNumColors() ) {
    case 2: // MONO
        modes = new int[] { 0, 1 };
        break;
    case 4: // GRAYSCALE
        modes = new int[] { 0, 1, 4, 5 };
        break;
    case 65536: // COLOR
        modes = new int[] { 0, 1, 8, 9, 11 };
        break;
}
```

Configure the pixel transparency in the drawing area

Pixel transparency is set in the current global alpha value for drawing operations. This value ranges from 0 (fully transparent) to 255 (fully opaque). The current value can be retrieved by using `Graphics.getGlobalAlpha()`, and can be set by using `Graphics.setGlobalAlpha()`.

1. Import the required class.

```
import net.rim.device.api.ui.Graphics;
```

2. Create an instance of a `Graphics` object.

```
private Graphics _g;
```

3. Perform one of the following tasks:
   - To set the global alpha value, invoke `_g.setGlobalApha()`.
   - To retrieve the global alpha value, invoke `_g.getGlobalApha()`.

Query the raster operations that the BlackBerry device application supports

Your BlackBerry® device application can use `Graphics.isRopSupported(introp)` to query if a particular raster operation is supported for the `Graphics` object.

1. Import the required class.

```
import net.rim.device.api.ui.Graphics;
```

2. Create an instance of a `Graphics` object.

```
private Graphics _g;
```

3. Invoke `Graphics.isRopSupported(int)` by passing a ROP_* or ROP2_* constant as the parameter.

```
if ( _g.isRopSupported(Graphics.ROP_SRC_COPY)) {
```

Draw a path that is shaded and filled

1. Import the required class.

```
import net.rim.device.api.ui.Graphics;
```

2. Invoke `Graphics.drawShadedFilledPath()`.

```
public void paint(Graphics g) {
    g.setColor(186895);
    int x[] = { 71,90,150,100,115,71,30,45,0,55 };
    int y[] = { 30,75,75,108,160,130,160,108,75,75 };
    int col[] =
{ 186895,186895,186895,186895,186895,186895,186895,186895,186895,186
    895 };

    g.drawShadedFilledPath( x, y, null, col, null );
    this.getUiEngine().updateDisplay();
}
```

Code sample: Drawing a path that blends from blue to red on the screen of a BlackBerry device application

```
int _width = Display.getWidth();
int _height = Display.getHeight();
Bitmap surface = new Bitmap(_width, _height);
BitmapField surfaceField = new BitmapField(surface);
add(surfaceField);
Graphics graphics = new Graphics(surface);
int[] X_PTS = { 0, 0, _width, _width };
int[] Y_PTS = { 20, 50, 50, 20 };
int[] drawColors = { 0x0000CC, 0x0000CC, 0xCC0000, 0xCC0000 };
try {
    graphics.drawShadedFilledPath(X_PTS, Y_PTS, null, drawColors, null);
} catch (IllegalArgumentException iae) {
    System.out.println("Bad arguments.");
}
```

Turn on or off a drawing style

1. Import the required class.

   ```
   import net.rim.device.api.ui.Graphics;
   ```

2. Create an instance of a `Graphics` object.

   ```
   private Graphics _g;
   ```

3. Invoke `Graphics.setDrawingStyle(int drawStyle, boolean on)` where the 'on' parameter is set to true to turn on a drawing style, and set to false to turn off a drawing style

   ```
   _g.setDrawingStyle(Graphics.DRAWSTYLE_AAPOLYGONS, true);
   _g.setDrawingStyle(Graphics.DRAWSTYLE_AAPOLYGONS, false);
   ```

Determine if a drawing style is configured

1. Import the required class.

   ```
   import net.rim.device.api.ui.Graphics;
   ```

2. Invoke `Graphics.isDrawingStyleSet(int drawStyle)`.

   ```
   if(g.isDrawingStyleSet(Graphics.DRAWSTYLE_FOCUS)) {
   ```

Code sample: Determining if the specified drawing style is turned on

```
public static void paintPanel( Graphics g, int width, int height,
     boolean isBottomRow, int panelType )
{
    int[] colors;
    if(g.isDrawingStyleSet(Graphics.DRAWSTYLE_FOCUS)) {
        colors = new int[]{
            COLOR_BACKGROUND_FOCUS,
            COLOR_BACKGROUND_FOCUS,
            COLOR_GRADIENT_END_FOCUS,
            COLOR_GRADIENT_END_FOCUS
        };
        g.setColor(COLOR_BACKGROUND_FOCUS);]
    } else {
        colors = new int[]{
            COLOR_BACKGROUND,
            COLOR_BACKGROUND,
            COLOR_GRADIENT_END,
            COLOR_GRADIENT_END
        };
        g.setColor(COLOR_BACKGROUND);
    }
}
```

Use a monochrome bitmap image as a stamp

Drawing a stamp on a bitmap image allows transparent pixels to remain transparent (only the opaque pixels are updated).

1. Import the required class.

   ```
   import net.rim.device.api.ui.component.BitmapField;
   ```

2. To use a monochrome bitmap image as a stamp by rendering the non-transparent region in color, create a `BitmapField` object using as a parameter the `STAMP_MONOCHROME` constant. Only use the `STAMP_MONOCHROME` constant with bitmap images that have a bit depth of 1 and have alpha defined.

   ```
   BitmapField field = new BitmapField(original,
        BitmapField.STAMP_MONOCHROME);
   ```

Code sample: Demonstrating use of a monochrome bitmap image as a stamp

```
final class DrawDemoScreen extends MainScreen {
    DrawDemoScreen() {
        LabelField title = new LabelField("UI Demo", LabelField.USE_ALL_WIDTH);
```

```
        setTitle(title);
        Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
        Bitmap restored = new Bitmap(original.getType(), original.getWidth(),
    original.getHeight());
        Graphics graphics = new Graphics(restored);

        // Retrieve raw data from original image.
        int[] argb = new int[original.getWidth() * original.getHeight()];

        original.getARGB(argb, 0, original.getWidth(), 0, 0, original.getWidth(),
    original.getHeight());

        // Draw new image using raw data retrieved from original image.
        graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0, restored.getWidth(),
    restored.getHeight());

        if(restored.equals(original)) {
            System.out.println("Success! Bitmap renders correctly with RGB data.");
        } else if(!restored.equals(original)) {
            System.out.println("Bitmap rendered incorrectly with RGB data.");
        }

        BitmapField field1 = new BitmapField(original,
    BitmapField.STAMP_MONOCHROME);
        BitmapField field2 = new BitmapField(restored);
        add(new LabelField("Original bitmap: "));
        add(field1);
        add(new LabelField("Restored bitmap: "));
        add(field2);
    }
}
```

Create a bitmap image using another bitmap image

A BlackBerry® device application can directly manipulate raw image data after the raw image data from a specified region of a bitmap image is stored in an integer array.

Create a bitmap image that contains the resized contents of another bitmap image

1. Import the required class.

   ```
   import net.rim.device.api.system.Bitmap;
   ```

2. Retrieve an existing bitmap image.

```
Bitmap testBitmap0 = Bitmap.getBitmapResource("rim.png");
```

3. Create a bitmap area to contain the scaled version of the original bitmap image.

```
Bitmap scaledBitmap1 = new Bitmap(200, 50);
```

4. Scale the source bitmap image and store it in another bitmap image.

```
testBitmap0.scaleInto(scaledBitmap1, Bitmap.FILTER_BILINEAR);
```

Duplicate a bitmap image

1. Import the required class.

```
import net.rim.device.api.system.Bitmap;
```

2. Add an existing bitmap image into a byte array.

```
byte[] data = getNamedResource("image.gif");
```

3. Create a new bitmap image that is an exact duplicate of the existing image.

```
Bitmap newBitmap = Bitmap.createBitmapFromBytes(data, 0, -1, 1);
```

Create a bitmap image that contains some of the content of another bitmap image

1. Import the required class.

```
import net.rim.device.api.system.Bitmap;
```

2. Load an existing bitmap image into a byte array.

```
byte[] data = getNamedResource("image.gif");
```

3. Create a new bitmap image that is a subset of the existing image where START_OFFSET is the starting offset into the byte array, and NUM_BYTES is the number of bytes to write to the new bitmap image.

```
Bitmap newBitmap = Bitmap.createBitmapFromBytes(data, START_OFFSET,
     NUM_BYTES, 1);
```

Draw an image on an empty bitmap image

1. Import the required classes.

```
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.component.BitmapField;
```

2. Create an empty bitmap image, copying the type and size from an existing bitmap image.

```
Bitmap restored = new Bitmap(original.getType(), original.getWidth(),
     original.getHeight());
```

3. Create a `Graphics` object and use the empty bitmap image as the drawing surface.

```
Graphics graphics = new Graphics(restored);
```

4. Invoke `Graphics.rawRGB()` to draw a new image using raw data retrieved from the original image.

```
graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0,
     restored.getWidth(), restored.getHeight());
```

Interpolation filters

Interpolation is required to eliminate a blocky appearance of a rescaled bitmap image due to changes in the number of pixels. Interpolation resamples bitmap image data and attempts to avoid a blocky pixelated appearance. Because the visual result of each filter can look different when applied to various types of images, it is up to the user to choose a suitable interpolation filter.

There are three filters available for use with bitmap image scaling methods.

Filter	Description
Bitmap.FILTER_LANCZOS	The Lanczos filter is the slowest of the available interpolation filters. Lanczos interpolation can produce sharp images, but might introduce some ringing artifacts.
Bitmap.FILTER_BOX	The box filter, also known as a mean filter, smoothing filter, and averaging filter, can quickly produce blurry images.
Bitmap.FILTER_BILINEAR	The bilinear filter can quickly produce good results for resized images, but might display sharp transition lines. This filter is often used to render bitmap image previews.

Displaying images

4

Displaying an image for zooming and panning

The `ZoomScreen` class allows you to provide zoom and pan the functionality to an image. When a BlackBerry® device user clicks the trackball or touch screen, the screen displays the center region of the image zoomed in.

When the image is zoomed in, the screen also displays an overlay highlighting the zoomed region. Scrolling the trackball or sliding a finger around the screen pans around the image. When the user stops panning and zooming the overlay disappears from the screen.

On BlackBerry devices with a touch screen, users can define the region of the image to zoom. The user can touch two points on the image to define the diagonally opposite corners of the region. When the region is selected, teh user can move one finger around the image to move the zoom region. To zoom in on the defined region, the user clicks the screen. To zoom out, the user presses the Escape key.

Display an image for zooming and panning

1. Import the required classes and interfaces:

```
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.extension.container.*;
```

2. Create the application framework by extending the `UiApplication` class. In `main()`, create an instance of the new class and invoke `enterEventDispatcher()` to enable the application to receive events. In the constructor, create an `EncodedImage` using a resource in your project, create a `ZoomScreen` with the `EncodedImage`, and invoke `pushScreen()` to display the image for zooming and panning.

```
public class ZoomableImageApp extends UiApplication
{
     public static void main(String[] args)
     {
          ZoomableImageApp theApp = new ZoomableImageApp();
          theApp.enterEventDispatcher();
     }

     public ZoomableImageApp()
     {
          EncodedImage myImg = EncodedImage.getEncodedImageResource("myImg.jpg");
          ZoomScreen zoomableImg = new ZoomScreen(myImg);
```

```
            pushScreen(zoomableImg);
        }
}
```

Code sample: Displaying an image for zooming and panning

```
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.extension.container.*;

public class ZoomableImageApp extends UiApplication
{
     public static void main(String[] args)
     {
          ZoomableImageApp theApp = new ZoomableImageApp();
          theApp.enterEventDispatcher();
     }

     public ZoomableImageApp()
     {
          EncodedImage myImg = EncodedImage.getEncodedImageResource("myImg.jpg");
          ZoomScreen zoomableImg = new ZoomScreen(myImg);
          pushScreen(zoomableImg);
     }
}
```

Displaying a row of images for scrolling

You can use the `PictureScrollField` class to render a horizontal row of images that the user can scroll through by using the trackball or touch gestures.

The `PictureScrollField` allows you to define the highlighting style of the selected image, the background style of the `PictureScrollField`, the text displayed below the selected image, the tooltip text that appears when an image is initially selected, and the height and width of the images.

Display a row of images for scrolling

1. Import the required classes and interfaces.

```
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
```

```
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.decor.*;
import net.rim.device.api.ui.extension.component.*;
```

2. Create the application framework by extending the `UiApplication` class. In `main()`, create an instance of the new class and invoke `enterEventDispatcher()` to enable the application to receive events. In the constructor, invoke `pushScreen()` to display the custom screen for the application. The `PictureScrollFieldDemoScreen` class, described in step 3, represents the custom screen.

```
public class PictureScrollFieldDemo extends UiApplication
{
     public static void main(String[] args)
     {
          PictureScrollFieldDemo app = new PictureScrollFieldDemo();
          app.enterEventDispatcher();
     }

     public PictureScrollFieldDemo()
     {
          pushScreen(new PictureScrollFieldDemoScreen());
     }
}
```

3. Create the framework for the custom screen by extending the `MainScreen` class.

```
class PictureScrollFieldDemoScreen extends MainScreen
{
     public PictureScrollFieldDemoScreen()
     {
     }
}
```

4. In the constructor, invoke `setTitle()` to set the text that appears in the title section of the screen.

```
setTitle("PictureScrollField Demo");
```

5. In the constructor, create and initialize three arrays to store the images to display in the `PictureScrollField` and the labels and tooltip text that appear when each image is selected. In this example, the `PictureScrollField` contains three images.

```
Bitmap[] images = new Bitmap[3];
String[] labels = new String[3];
String[] tooltips = new String[3];

images[0] = Bitmap.getBitmapResource("img1.jpg");
labels[0] = "Label for image 1";
tooltips[0] = "Tooltip for image 1";

images[1] = Bitmap.getBitmapResource("img2.jpg");
labels[1] = "Label for image 2";
tooltips[1] = "Tooltip for image 2";
```

```
images[2] = Bitmap.getBitmapResource("img3.jpg");
labels[2] = "Label for image 3";
tooltips[2] = "Tooltip for image 3";
```

6. In the constructor, create and initialize an array of `ScrollEntry` objects. A `ScrollEntry` represents each item in the `PictureScrollField`.

```
ScrollEntry[] entries =  ScrollEntry[3];

for (int i = 0; i < entries.length; i++)
{
     entries[i] = new ScrollEntry(images[i], labels[i], tooltips[i]);
}
```

7. In the constructor, create and initialize the `PictureScrollField` with each image 150 pixels wide and 100 pixels high. Invoke `setData()` to set the entries in the `PictureScrollField`. The second paramater in `setData()` specifies which image position is selected by default.

```
PictureScrollField pictureScrollField = new PictureScrollField(150,
     100);
pictureScrollField.setData(entries, 0);
```

8. In the constructor, set the style of the `PictureScrollField`. Invoke `setHighlightStyle()` to specify how the selected image is highlighted. Invoke `setHighlightBorderColor()` to specify the selected image's border color. Invoke `setBackground()` to specify the background of the `PictureScrollField`. Invoke `setLabelVisible()` to specify whether the `PictureScrollField` displays the selected image's label.

```
pictureScrollField.setHighlightStyle(HighlightStyle.ILLUMINATE);
pictureScrollField.setHighlightBorderColor(Color.BLUE);
pictureScrollField.setBackground(BackgroundFactory
    .createSolidTransparentBackground(Color.RED, 150));
pictureScrollField.setLabelsVisible(true);
```

9. In the constructor, invoke `add()` to display the `PictureScrollField`.

```
add(pictureScrollField);
```

Code sample: Displaying a row of images for scrolling

```
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.decor.*;
import net.rim.device.api.ui.extension.component.*;

public class PictureScrollFieldDemo extends UiApplication
{
     public static void main(String[] args)
     {
```

```
          PictureScrollFieldDemo app = new PictureScrollFieldDemo();
          app.enterEventDispatcher();
     }

     public PictureScrollFieldDemo()
     {
          pushScreen(new PictureScrollFieldDemoScreen());
     }
}

class PictureScrollFieldDemoScreen extends MainScreen
{
     public PictureScrollFieldDemoScreen()
     {
          setTitle("PictureScrollField Demo");

          Bitmap[] images = new Bitmap[3];
          String[] labels = new String[3];
          String[] tooltips = new String[3];

          images[0] = Bitmap.getBitmapResource("img1.jpg");
          labels[0] = "Label for image 1";
          tooltips[0] = "Tooltip for image 1";

          images[1] = Bitmap.getBitmapResource("img2.jpg");
          labels[1] = "Label for image 2";
          tooltips[1] = "Tooltip for image 2";

          images[2] = Bitmap.getBitmapResource("img3.jpg");
          labels[2] = "Label for image 3";
          tooltips[2] = "Tooltip for image 3";

          ScrollEntry[] entries =  ScrollEntry[3];

          for (int i = 0; i < entries.length; i++)
          {
               entries[i] = new ScrollEntry(images[i], labels[i],tooltips[i]);
          }


          PictureScrollField pictureScrollField = new PictureScrollField(150, 100);
          pictureScrollField.setData(entries, 0);
          pictureScrollField.setHighlightStyle(HighlightStyle.ILLUMINATE);
          pictureScrollField.setHighlightBorderColor(Color.BLUE);

          pictureScrollField.setBackground
(BackgroundFactory.createSolidTransparentBackground(Color.RED, 150));

          pictureScrollField.setLabelsVisible(true);
```

```
            add(pictureScrollField);
        }
}
```

Using unprocessed images

5

Retrieve unprocessed image data

1. Import the required class.

```
import net.rim.device.api.system.Bitmap;
```

2. Invoke `Bitmap.getARGB()` to retrieve the unprocessed image data from a region of a bitmap image and store the data in an integer array.

```
Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
int[] argb = new int[original.getWidth() * original.getHeight()];
```

Store unprocessed image data

1. Import the required class.

```
import net.rim.device.api.system.Bitmap;
```

2. Create a `Bitmap` object from an existing bitmap image.

```
Bitmap img = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
```

3. Initialize an integer array.

```
int[] argb = new int[img.getWidth() * img.getHeight()];
```

4. Invoke Bitmap.getARGB() to store the unprocessed image data from a new or pre-defined bitmap in an integer array.

```
img.getARGB(argb, 0, img.getWidth(), 0, 0, img.getWidth(),
     img.getHeight());
```

Compare two unprocessed images

1. Import the required class.

```
import net.rim.device.api.system.Bitmap;
```

2. Invoke `Bitmap.equals()`.

```
if(restored.equals(original)) {
    System.out.println("Success! Bitmap renders correctly with RGB data.");
} else (!restored.equals(original)) {
    System.out.println("Bitmap rendered incorrectly with RGB data.");
}
```

Compare two bitmap images to check if they are different

1. Import the required class.

```
import net.rim.device.api.system.Bitmap;
```

2. Create two Bitmap objects from bitmap images to compare.

```
Bitmap _first = Bitmap.getBitmapResource("first.png");
Bitmap _last  = Bitmap.getBitmapResource("last.png");
```

3. Create an `XYRect` object to reflect the differences between bit images.

```
XYRect diff = new XYRect(0, 0, _first.getWidth(),
     _first.getHeight());
```

4. Compare the bitmap images by using the `Bitmap.locateDifference(XYRect diffRect, Bitmap bmp2, int offset_x, int offset_y)` method. This comparison is done within the provided `XYRect` reference and can be offset within the second bitmap image to an alternate position. The minimal difference between the bitmap images in the defined area is determined, with the `XYRect` reference modified with the new coordinates representing this area. An empty `XYRect` represents no difference between the two bitmaps within the given area.

```
_first.locateDifference( diff, _last, 0, 0 );
```

Using encoded images

6

Access an encoded image through an input stream

1. Import the required classes.

```
import java.io.InputStream;
```

2. Save the image to the project folder or sub-folder.
3. Add the image to the project in the BlackBerry® Java® Plug-in Eclipse® or the BlackBerry® Java® Development Environment.
4. Invoke `getClass().getResourceAsStream()` to retrieve the image as an input stream of bytes.

```
private InputStream input;
...
Class _class = this.getClass();
input = _class.getResourceAsStream("/images/example.png");
```

Encode an image

1. Import the required classes.

```
import net.rim.device.api.system.EncodedImage;
```

2. Invoke `EncodedImage.createEncodedImage()`. This method uses the unprocessed image data in the byte array to create an instance of `EncodedImage`.
3. Check for an `IllegalArgumentException`, which `EncodedImage.createEncodedImage()` throws if the byte array that you provide as a parameter does not contain a recognized image format.

```
// Store the contents of the image file.
private byte[] data = new byte[2430];
try {
    // Read the image data into the byte array
    input.read(data);
} catch (IOException e) {
    // Handle exception.
}
try {
    EncodedImage image = EncodedImage.createEncodedImage(data, 0, data.length);
} catch (IllegalArgumentException iae) {
    System.out.println("Image format not recognized.");
}
```

Display an encoded image

1. Import the required class.

```
import net.rim.device.api.ui.component.BitmapField;
```

2. Invoke `BitmapField.setImage()` to assign the encoded image to a `BitmapField`.
3. Invoke `add()` to add the `BitmapField` to the screen.

```
BitmapField field = new BitmapField();
field.setImage(image);
add(field);
```

Specify the decoding mode for an image

1. Import the required class.

```
import net.rim.device.api.system.EncodedImage;
```

2. Invoke `EncodedImage.setDecodeMode()` using one of the following modes as a parameter:
   - Use `DECODE_ALPHA` to decode an alpha channel, if one exists (this is the default mode).
   - Use `DECODE_NATIVE` to force decoding of the bitmap image to the native bitmap image type of the BlackBerry® Device Software.
   - Use `DECODE_READONLY` to mark the decoded bitmap image as read-only.

Specify the display size of an encoded image

1. Import the required class.

```
import net.rim.device.api.system.EncodedImage;
```

2. Invoke `EncodedImage.scaleImage32(int scaleX, int scaleY)`. The passed scale value parameters must be `net.rim.device.api.math.Fixed32` numbers (0 < scale < 1 for upscaling, and scale > 1 for downscaling). `scaleImage32()` creates a new `EncodedImage` instead of modifying the current one.

Using SVG content in a BlackBerry device application 7

The BlackBerry® device supports a binary representation of Scalable Vector Graphics (SVG) content in the form of a .pme file.

You can use the BlackBerry® Composer to create a .pme file or use the Plazmic SVG Transcoding Utility to convert an .svg file to a .pme file. The BlackBerry Composer and the Plazmic SVG Transcoding Utility are included in the BlackBerry® Theme Studio.

Download SVG content

1. Import the required class.

```
import net.rim.plazmic.mediaengine.MediaManager;
```

2. Create a `MediaManager` object.

```
MediaManager manager = new MediaManager();
```

3. Invoke `MediaManager.createMedia()`. The first time that you invoke `MediaManager.createMedia()`, the URL must be absolute, unless you first invoke `MediaManager.setProperty("URI_BASE", <base_url>)` to set a base URL. When you invoke `createMedia()` subsequently, the previous URL is used as the base.

```
try {
    Object media = manager.createMedia("http://webserver/sample.pme");
} catch (IOException ioe) {
    System.out.println("Error: requested content was not downloaded.");
} catch (MediaException me) {
    System.out.println("Error: " + me.getCode());
}
```

Play rich media content

1. Import the required classes.

```
import net.rim.plazmic.mediaengine.MediaPlayer;
import net.rim.plazmic.mediaengine.MediaManager;
```

2. Create a class that extends `MainScreen`, and a constructor.

```
final static class MediaSampleScreen extends MainScreen {
    public MediaSampleScreen() {
        super();
        LabelField title = new LabelField("Media Sample", LabelField.ELLIPSIS |
     LabelField.USE_ALL_WIDTH);
        setTitle(title);
```

3. Create a `MediaPlayer` object to handle playback of the media types supported by the Media Engine.

```
MediaPlayer player = new MediaPlayer();
```

4. Create a `MediaManager` object to handle downloading of media content from the Internet or local storage.

```
MediaManager manager = new MediaManager();
```

5. Invoke `MediaManager.createMedia(String)` to retrieve media content from a given URI.

```
Object media = manager.createMedia("http://webserver/SVGFILE.pme");
```

6. Invoke `MediaPlayer.setMedia()` to specify the media for playback.

```
player.setMedia(media);
```

7. Invoke `MediaPlayer.getUI()` to retrieve a UI object that displays PME content. Cast the resulting object as a `Field`, and then `add` it to a `Screen` for display.

```
add((Field)player.getUI());
```

8. Invoke `MediaPlayer.start()` to start playing the downloaded PME content.

```
player.start();
```

After you finish: You must create a screen that does not support scrolling to display SVG content from the Plazmic® Media Engine version 4.2.

Listen for events while downloading a .pme file

1. Import the required class and interface.

```
import net.rim.plazmic.mediaengine.MediaManager;
import net.rim.plazmic.mediaengine.MediaListener;
```

2. Implement the `net.rim.plazmic.mediaengine.MediaListener` interface to handle all possible media events.

```
public final class MediaListenerImpl implements MediaListener {
    public void mediaEvent(Object sender, int event, int eventParam, Object data) {
        switch(event) {
            case MEDIA_REQUESTED:
                break;
            case MEDIA_COMPLETE:
                break;
            case MEDIA_REALIZED:
                break;
            case MEDIA_IO:
                break;
        }
    }
}
```

3. Invoke `addMediaListener()` on the `MediaPlayer` and `MediaManager` objects to register the listener for media events.

```
private MediaListenerImpl _listener = new MediaListenerImpl();
private MediaPlayer player = new MediaPlayer();
private MediaManager manager = new MediaManager();
player.addMediaListener(_listener);
manager.addMediaListener(_listener);
```

Respond to events while downloading a .pme file

1. Import the required class and interface.

```
import net.rim.plazmic.mediaengine.io.LoadingStatus;
import net.rim.plazmic.mediaengine.MediaListener;
```

2. Implement the `net.rim.plazmic.mediaengine.MediaListener` interface.

```
public final class MediaListenerImpl implements MediaListener {
```

3. In your implementation of `MediaListener.mediaEvent()`, when the `MEDIA_IO` event occurs, cast the `Object` in the data parameter to a `LoadingStatus` object.

```
LoadingStatus s = (LoadingStatus)data;
```

4. Invoke `LoadingStatus.getStatus()` to download content and manage the status of the .pme file that the application is downloading.

```
switch(s.getStatus()) {
```

5. For each normal status, print a message to the console.

```
case LoadingStatus.LOADING_STARTED:
    System.out.println("Loading in progress");
    break;

case LoadingStatus.LOADING_READING:
    System.out.println("Parsing in progress");
    break;

case LoadingStatus.LOADING_FINISHED:
    System.out.println("Loading completed");
    break;
```

6. If `LoadingStatus.getStatus()` returns a value equal to `LoadingStatus.LOADING_FAILED`, perform one or more of the following actions:
   - To retrieve the error code, invoke `LoadingStatus.getCode()`.
   - To retrieve the detailed message, invoke `LoadingStatus.getMessage()`.
   - To retrieve the URL string of the content, invoke `LoadingStatus.getSource()`.

```
case LoadingStatus.LOADING_FAILED:
    String errorName = null;
    int code = s.getCode();

    switch (code)
    {
        case MediaException.INVALID_HEADER:
            errorName = "Invalid header"+ "/n" + s.getSource();
            break;

        case MediaException.REQUEST_TIMED_OUT:
            errorName = "Request timed out" + "\n" + s.getSource();
            break;

        case MediaException.INTERRUPTED_DOWNLOAD:
            break;

        case MediaException.UNSUPPORTED_TYPE:
           errorName = "Unsupported type" + s.getMessage() + "\n" + s.getSource();
            break;

        default:
            {
                if (code > 200) {
                    // A code > 200 indicates an HTTP error.
                    errorName = "URL not found";
                } else {
                    // Default unidentified error.
                    errorName = "Loading Failed";
                }
                errorName += "\n" + s.getSource() + "\n" + s.getCode() + ": " +
 s.getMessage();
                break;
            }
    }
    System.out.println(errorName);
    break;
```

Code sample: Responding to events when a BlackBerry device application downloads SVG content

```
package com.rim.samples.device.mediaenginedemo;

import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

/**
 * The MediaListener implementation.
```

```
 */
public final class MediaListenerImpl implements MediaListener
{
    public void mediaEvent(Object sender, int event, int eventParam, Object data)
    {
        switch(event)
        {
            case MEDIA_REQUESTED:
                System.out.println("Media requested");
                break;

            case MEDIA_COMPLETE:
                System.out.println("Media completed");
                break;

            case MEDIA_REALIZED:
                System.out.println("Media realized");
                break;

            case MEDIA_IO:
            {
                LoadingStatus s = (LoadingStatus)data;

                switch(s.getStatus())
                {
                    case LoadingStatus.LOADING_STARTED:
                        System.out.println("Loading in progress");
                        break;

                    case LoadingStatus.LOADING_READING:
                        System.out.println("Parsing in progress");
                        break;

                    case LoadingStatus.LOADING_FINISHED:
                        System.out.println("Loading completed");
                        break;

                    case LoadingStatus.LOADING_FAILED:
                        String errorName = null;
                        int code = s.getCode();

                        switch (code)
                        {
                            case MediaException.INVALID_HEADER:
                                errorName = "nvalid header"+ "/n" + s.getSource();
                                break;

                            case MediaException.REQUEST_TIMED_OUT:
                                errorName = "Request timed out" + "\n" +
     s.getSource();
                                break;
```

```
                            case MediaException.INTERRUPTED_DOWNLOAD:
                                break;

                            case MediaException.UNSUPPORTED_TYPE:
                                errorName = "Unsupported type" + s.getMessage() +
    "\n" + s.getSource();
                                break;

                            default:
                            {
                                if (code > 200)
                                {
                                    // A code > 200 indicates an HTTP error.
                                    errorName = "URL not found";
                                }
                                else
                                {
                                    // Default unidentified error.
                                    errorName = "Loading Failed";
                                }
                                    errorName += "\n" + s.getSource() + "\n"
                                    + s.getCode() + ": " + s.getMessage();
                                    break;
                            }
                        }

                        System.out.println(errorName);
                        break;

                } // End switch s.getStatus().

                break;
            }
        }
    }
}
```

Download and play a .pme file

1. Import the required classes and interface.

   ```
   import net.rim.plazmic.mediaengine.MediaPlayer;
   import net.rim.plazmic.mediaengine.MediaManager;
   import net.rim.plazmic.mediaengine.MediaListener;
   ```

2. Implement the `net.rim.plazmic.mediaengine.MediaListener` interface.

   ```
   public final class MediaListenerImpl implements MediaListener {
   ```

3. Create a MediaManager object and then invoke `MediaManager.createMediaLater()`.

```
MediaManager manager = new MediaManager();
manager.createMediaLater("http://webserver/sample.pme");
```

4. In `MediaListener.mediaEvent()` method, add code to manage the `MEDIA_REALIZED` event that occurs when the application finishes loading a .pme file to the BlackBerry® device.

```
public void mediaEvent(Object sender, int event, int eventParam,
     Object data) {
    switch(event) {
        ...
        case MEDIA_REALIZED:
```

5. Invoke `MediaPlayer.setMedia(data)` to register the content.

```
player.setMedia(data);
```

6. Invoke `MediaPlayer.start()` to start playing the content.

```
player.start();
```

Play SVG content

1. Import the required classes.

```
import net.rim.device.api.ui.Screen;
import net.rim.plazmic.mediaengine.MediaPlayer;
import net.rim.plazmic.mediaengine.MediaException;
```

2. Create a `MediaPlayer` object to handle playback of the media types supported by the Media Engine.

```
MediaPlayer player = new MediaPlayer();
```

3. Invoke `MediaPlayer.setMedia()` to prepare the `MediaPlayer` to play media.

```
try {
    player.setMedia(media);
} catch (MediaException me) {
    System.out.println("Error: requested content type is not supported.");
}
```

4. Create an instance of a screen object using the `NO_VERTICAL_SCROLL` and `NO_HORIZONTAL_SCROLL` fields to allow a screen to display a .pme file that the Plazmic® Media Engine version 4.2 creates.

```
Screen screen = new Screen(Screen.NO_VERTICAL_SCROLL |
     Screen.NO_HORIZONTAL_SCROLL);
```

5. Invoke `MediaPlayer.getUI()`, cast the returned object to a `Field` type, and `add` it to a `Screen` to display.

```
screen.add((Field)player.getUI());
```

6. If the `Player` object is in the `REALIZED` state, invoke `MediaPlayer.start()` to start playing the .pme file.

```
if(player.getState() == MediaPlayer.REALIZED) {
    try {
        player.start();
    } catch(MediaException me) {
       System.out.println("Error occurred during media playback: " + me.getCode() +
     me.getMessage());
    }
}
```

Access SVG content through a connection that MediaManager does not support

You can create a BlackBerry® device application that can download SVG content from the Internet or from local storage using a connection that supports a custom protocol or has functionality that the MediaManager does not support or provide.

1. Import the required classes.

```
import java.io.*;
import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;
```

2. To implement a custom connector, implement the `net.rim.plazmic.mediaengine.io.Connector` interface.

```
public class CustomPMEConnector implements Connector {
    private Connector delegate;
    private InputStream input;

    CustomPMEConnector(Connector delegate) {
        this.delegate = delegate;
    }
    public InputStream getInputStream(String uri, ConnectionInfo info)
        throws IOException, MediaException {
        if (uri.startsWith("myprotocol://")) {
            // Perform special tasks.
            info.setConnection(new MyProtocolConnection());
            info.setContentType("application/x-vnd.rim.pme");
            // OpenMyInputStream() is a custom method that opens
            //stream for "myprotocol://"
            input = openMyInputStream(uri);
        } else {
            input = delegate.getInputStream(uri, info);
        }
        return input;
    }

    private InputStream openMyInputStream(String uri) {
        InputStream input = null;
        // @todo: open stream here
        return input;
    }
```

```
    // Inner class that defines the connection class.
    public static class MyProtocolConnection {
        public MyProtocolConnection() {
            // ...
        }
        public void close() {
            // ...
        }
    }
```

3. To configure custom connector properties, implement `setProperty(String, String)`.

```
public void setProperty(String property, String value) {
    delegate.setProperty(property, value);
}
```

4. To release the custom connection, implement `releaseConnection(ConnectionInfo)`.

```
public void releaseConnection(ConnectionInfo info) throws
     IOException, MediaException {
    Object o = info.getConnection();
    if (o instanceof MyProtocolConnection) {
        ((MyProtocolConnection)o).close(); // Perform cleanup.
    } else {
        delegate.releaseConnection(info);
    }
}
```

Code sample: Implementing a custom connector framework.

```
/*
 * CustomPMEConnector.java
 * Copyright (C) 2003-2007 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

public class CustomPMEConnector implements Connector {
    private Connector delegate;
    private InputStream input;

    CustomPMEConnector(Connector delegate) {
        this.delegate = delegate;
    }

    public InputStream getInputStream(String uri, ConnectionInfo info) throws
     IOException, MediaException {
```

```
        if (uri.startsWith("myprotocol://")) {
            // Perform special tasks.
            info.setConnection(new MyProtocolConnection());
            info.setContentType("application/x-vnd.rim.pme");
            // OpenMyInputStream() is a custom method that opens
            //stream for "myprotocol://"
            input = openMyInputStream(uri);
        } else {
            input = delegate.getInputStream(uri, info);
        }
        return input;
    }

    private InputStream openMyInputStream(String uri) {
        InputStream input = null;
        // @todo: open stream here
        return input;
    }

    public void releaseConnection(ConnectionInfo info)
        throws IOException, MediaException {
        Object o = info.getConnection();
        if (o instanceof MyProtocolConnection) {
            ((MyProtocolConnection)o).close(); // Perform cleanup.
        } else {
            delegate.releaseConnection(info);
        }
    }

    public void setProperty(String property, String value) {
        delegate.setProperty(property, value);
    }

    // Inner class that defines the connection class.
    public static class MyProtocolConnection {
        public MyProtocolConnection() {
            // ...
        }
        public void close() {
            // ...
        }
    }
}
```

Creating 2-D and 3-D graphics by using JSR-239

8

You can create 2-D and 3-D graphics for a BlackBerry® device by using JSR-239 and BlackBerry APIs. JSR-239 contains the Java® binding for OpenGL® ES. OpenGL ES is based on the desktop version of OpenGL, but has been designed for use on wireless devices.

For more information about OpenGL, visit www.khronos.org/opengles/.

Packages for JSR-239 support

Package	Description
`javax.microedition.khronos.opengles`	This package provides a subset of OpenGL APIs and the bindings for OpenGL® ES 1.0 and their extensions. It includes methods for floating and fixed-point manipulation of values.
`javax.microedition.khronos.egl`	This package provides the bindings for EGL™ 1.0 for the OpenGL ES API. The rendering contexts that are supported can provide a container for the OpenGL ES rendering state. The package is designed to support Window, Pbuffer, and Pixmap rendering surfaces.
`net.rim.device.api.opengles`	The `GLUtils` class in this package provides support for following actions: • verifying that the BlackBerry® device supports OpenGL ES • loading a 3-D perspective projection matrix • loading a view transformation matrix • defining a 2-D orthographic projection matrix • loading a texture from a `Bitmap` image • freeing memory that is currently allocated for a direct buffer
`java.nio`	This package provides the classes that enable your application to read from and write to high-speed block I/O.
`net.rim.device.api.math`	This package provides the classes that are designed to help you enhance the arithmetic versatility and processing speed of the BlackBerry device applications that you create.

Code sample: Rendering a multicolor 2-D triangle

The following code sample to demonstrates how to use JSR-239 to render a multicolor 2-D triangle.

```
import java.nio.*;

import javax.microedition.khronos.egl.*;
import javax.microedition.khronos.opengles.*;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.opengles.*;

public final class OpenGLTest extends UiApplication
{
    public OpenGLTest()
    {
        pushScreen(new OpenGLTestScreen());
    }

    public static void main(String[] args)
    {
        new OpenGLTest().enterEventDispatcher();
    }
}

class OpenGLTestScreen extends FullScreen implements Runnable
{
    private EGL11 _egl;
    private EGLDisplay _eglDisplay;
    private EGLConfig _eglConfig;
    private EGLSurface _eglSurface;
    private EGLContext _eglContext;
    private GL10 _gl;

    private Bitmap _offscreenBitmap;
    private Graphics _offscreenGraphics;

    private FloatBuffer _vertexArray;
    private FloatBuffer _colorArray;

    private boolean _running;
    private boolean _paused;

    OpenGLTestScreen()
    {
        super(FullScreen.DEFAULT_MENU | FullScreen.DEFAULT_CLOSE);
```

```
    }

    private void initialize()
    {
        // Get EGL interface
        _egl = (EGL11)EGLContext.getEGL();

        // Get the EGL display
        _eglDisplay = _egl.eglGetDisplay(EGL11.EGL_DEFAULT_DISPLAY);

        // Initialize the display for EGL setting the version to null
        _egl.eglInitialize(_eglDisplay, null);

        // Choose an EGL config
        EGLConfig[] configs = new EGLConfig[1];
        int[] numConfigs = new int[1];
        int[] attrs =
        {
            EGL11.EGL_RED_SIZE,   5,
            EGL11.EGL_GREEN_SIZE, 6,
            EGL11.EGL_BLUE_SIZE,  5,
            EGL11.EGL_NONE
        };
        _egl.eglChooseConfig(_eglDisplay, attrs, configs, 1, numConfigs);
        _eglConfig = configs[0];

        // Create an EGL window surface
        _eglSurface = _egl.eglCreateWindowSurface
            (_eglDisplay, _eglConfig, this, null);

        // Create an EGL context
        createEGLContext();

        // Specify vertices and colors for a triangle
        float[] vertices =
        {
            -0.5f, -0.5f, 0.0f,
            0.0f, 0.5f, 0.0f,
            0.5f, -0.5f, 0.0f
        };
        float[] colors =
        {
            0.0f, 1.0f, 0.0f, 1.0f,
            1.0f, 0.0f, 0.0f, 1.0f,
            0.0f, 0.0f, 1.0f, 1.0f
        };
        _vertexArray = ByteBuffer.allocateDirect(3 * 3 * 4).asFloatBuffer();
        _vertexArray.put(vertices);
        _vertexArray.rewind();
        _colorArray = ByteBuffer.allocateDirect(4 * 3 * 4).asFloatBuffer();
        _colorArray.put(colors);
        _colorArray.rewind();
```

```
    }

    private void createEGLContext()
    {
        // Create an EGL context
        _eglContext = _egl.eglCreateContext
            (_eglDisplay, _eglConfig, EGL10.EGL_NO_CONTEXT, null);

        // Get the GL interface for the new context
        _gl = (GL10)_eglContext.getGL();

        // Make the new context current
        _egl.eglMakeCurrent
           (_eglDisplay, _eglSurface, _eglSurface, _eglContext);
    }

    private void destroyEGL()
    {
        _egl.eglMakeCurrent(_eglDisplay, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_CONTEXT);
        _egl.eglDestroyContext(_eglDisplay, _eglContext);
        _egl.eglDestroySurface(_eglDisplay, _eglSurface);
    }

    private void handleContextLost()
    {
        // Destroy the EGL context
        _egl.eglMakeCurrent(_eglDisplay, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_CONTEXT);
        _egl.eglDestroyContext(_eglDisplay, _eglContext);
        _eglContext = EGL10.EGL_NO_CONTEXT;

        // Re-create the EGL context
        createEGLContext();
    }

    /**
     * Main render loop.
     */
    public void run()
    {
        initialize();

        while (_running)
        {
            // Idle if this thread is in the background
            if (_paused)
            {
                synchronized (this)
                {
                    try
                    {
```

```
                    wait();
                }
                catch (InterruptedException x) { }
            }
        }

        updateBackBuffer();

        renderFrame();

        // Throttle cpu usage
        try
        {
            Thread.sleep(20);
        }
        catch (InterruptedException x) { }
    }

    destroyEGL();
}

private void renderFrame()
{
    // Make the context and surface current and check for EGL_CONTEXT_LOST
    if (!_egl.eglMakeCurrent(_eglDisplay, _eglSurface, _eglSurface,
 _eglContext))
    {
        if (_egl.eglGetError() == EGL11.EGL_CONTEXT_LOST)
            handleContextLost();
    }

    // Signal that OpenGL rendering is about to begin
    _egl.eglWaitNative(EGL10.EGL_CORE_NATIVE_ENGINE, _offscreenGraphics);

    render(_gl);

    // Signal that OpenGL ES rendering is complete
    _egl.eglWaitGL();

    // Swap the window surface to the display
    _egl.eglSwapBuffers(_eglDisplay, _eglSurface);
}

private void render(GL10 gl)
{
    // Set the GL viewport
    gl.glViewport(0, 0, getWidth(), getHeight());

    // Clear the surface
    gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
```

```
        // Set the projection matrix
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        GLUtils.gluPerspective
            (gl, 45.0f, (float)getWidth()/(float)getHeight(), 0.15f, 10.0f);

        // Draw the triangle
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0.0f, 0.0f, -3.0f);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, _vertexArray);
        gl.glColorPointer(4, GL10.GL_FLOAT, 0, _colorArray);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
    }

    /**
     * Called by the UI system to paint the screen.
     */
    protected void paint(Graphics g)
    {
        if (_offscreenBitmap != null)
            g.drawBitmap(0, 0, _offscreenBitmap.getWidth(),
                _offscreenBitmap.getHeight(), _offscreenBitmap, 0, 0);
    }

    /**
     * Called when the visibility of the screen changes.
     *
     * @param visible true if the screen is being made visible,
     * false if hidden
     */
    protected void onVisibilityChange(boolean visible)
    {
        if (visible)
        {
            resume();
        }
        else
        {
            pause();
        }
    }

    /**
     * Called when the screen is closing.
     */
    public void close()
    {
        _running = false;
        synchronized (this) { notifyAll(); }
```

```
        super.close();
    }

    /**
     * Keeps the back buffer in sync with the screen size.
     */
    private void updateBackBuffer()
    {
        if (_offscreenBitmap != null)
        {
            if (_offscreenBitmap.getWidth() == getWidth() &&
                _offscreenBitmap.getHeight() == getHeight())
                return; // no change needed
        }

        _offscreenBitmap = new Bitmap(getWidth(), getHeight());
        _offscreenGraphics = Graphics.create(_offscreenBitmap);
    }

    private void pause()
    {
        _paused = true;
    }

    private void resume()
    {
        if (_running)
        {
            // Pause the render loop
            _paused = false;
            synchronized (this) { notifyAll(); }
        }
        else
        {
            // Start the render thread.
            _running = true;
            new Thread(this).start();
        }
    }
}
```

Code sample: Drawing a 3-D cube

You can use the `GL10` interface to draw a 3-D cube.

```
import java.nio.ByteBuffer;
import java.nio.FloatBuffer;
```

```
import javax.microedition.khronos.egl.EGL10;
import javax.microedition.khronos.egl.EGL11;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.egl.EGLContext;
import javax.microedition.khronos.egl.EGLDisplay;
import javax.microedition.khronos.egl.EGLSurface;
import javax.microedition.khronos.opengles.GL10;

import net.rim.device.api.system.Bitmap;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.container.FullScreen;

class ThreeDCube extends FullScreen implements Runnable
{
    private EGL11 _egl;
    private EGLDisplay _eglDisplay;
    private EGLConfig _eglConfig;
    private EGLSurface _eglSurface;
    private EGLContext _eglContext;
    private GL10 _gl;

    private Bitmap _offscreenBitmap;
    private Graphics _offscreenGraphics;


    private boolean _running;
    private boolean _paused;
    private FloatBuffer _cubeVertices, _cubeNormals, _cubeColors;

    float _angle = 45f;

    private int _vertexCount;

    private static final float[] _vertices =
    {
        // front
        -0.5f, 0.5f, 0.5f, -0.5f, -0.5f, 0.5f, 0.5f, 0.5f, 0.5f,
        0.5f, 0.5f, 0.5f, -0.5f, -0.5f, 0.5f, 0.5f, -0.5f, 0.5f,

        // right
        0.5f, 0.5f, 0.5f, 0.5f, -0.5f, 0.5f, 0.5f, 0.5f, -0.5f,
        0.5f, 0.5f, -0.5f, 0.5f, -0.5f, 0.5f, 0.5f, -0.5f, -0.5f,

        // back
        0.5f, 0.5f, -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, 0.5f, -0.5f,
        -0.5f, 0.5f, -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, -0.5f, -0.5f,

        // left
        -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, -0.5f, -0.5f, 0.5f, 0.5f,
        -0.5f, 0.5f, 0.5f, -0.5f, -0.5f, -0.5f, -0.5f, -0.5f, 0.5f,

        // top
```

```
        -0.5f, 0.5f, -0.5f, -0.5f, 0.5f, 0.5f, 0.5f, 0.5f, -0.5f,
        0.5f, 0.5f, -0.5f, -0.5f, 0.5f, 0.5f, 0.5f, 0.5f, 0.5f,

        // bottom
        -0.5f, -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, 0.5f, -0.5f, 0.5f,
        0.5f, -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, 0.5f, -0.5f, -0.5f
    };

    private static final float[] _normals =
    {
        /* front */ 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
        /* right */ 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
        /* back */ 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1,
        /* left */ -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0,
        /* top */ 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
        /* bottom */ 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0
    };

    private static final float[] _colors =
    {
        /* front  - white  */  1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1,
     1,1,1,1,
        /* right  - red    */  1,0,0,1, 1,0,0,1, 1,0,0,1, 1,0,0,1, 1,0,0,1,
     1,0,0,1,
        /* back   - green  */  0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1,
     0,1,0,1,
        /* left   - blue   */  0,0,1,1, 0,0,1,1, 0,0,1,1, 0,0,1,1, 0,0,1,1,
     0,0,1,1,
        /* top    - yellow */  1,1,0,1, 1,1,0,1, 1,1,0,1, 1,1,0,1, 1,1,0,1, 1,1,0,1,
        /* bottom - magenta*/  1,0,1,1, 1,0,1,1, 1,0,1,1, 1,0,1,1, 1,0,1,1, 1,0,1,1
    };


    ThreeDCube()
    {
        super(FullScreen.DEFAULT_MENU | FullScreen.DEFAULT_CLOSE);
    }

    private void initialize()
    {
        // Get EGL interface
        _egl = (EGL11)EGLContext.getEGL();

        // Get the EGL display
        _eglDisplay = _egl.eglGetDisplay(EGL11.EGL_DEFAULT_DISPLAY);

        // Initialize the display for EGL, null since we don't really need the
//  version.
        _egl.eglInitialize(_eglDisplay, null);

        // Choose an EGL config
        EGLConfig[] configs = new EGLConfig[1];
```

```
        int[] numConfigs = new int[1];
        int[] attrs =
        {
            EGL11.EGL_RED_SIZE,   5,
            EGL11.EGL_GREEN_SIZE, 6,
            EGL11.EGL_BLUE_SIZE,  5,
            EGL11.EGL_NONE
        };
        _egl.eglChooseConfig(_eglDisplay, attrs, configs, 1, numConfigs);
        _eglConfig = configs[0];

        // Create an EGL window surface
        _eglSurface = _egl.eglCreateWindowSurface
            (_eglDisplay, _eglConfig, this, null);

        // Create an EGL context
        createEGLContext();

        _cubeVertices = createVertexBuffer();
        _cubeNormals = createNormalBuffer();
        _cubeColors = createColorBuffer();

        _vertexCount = _vertices.length / 3;
    }

    private FloatBuffer createVertexBuffer()
    {
        FloatBuffer buffer = ByteBuffer.allocateDirect(_vertices.length *
     4).asFloatBuffer();
        buffer.put(_vertices);
        buffer.rewind();
        return buffer;
    }

    private FloatBuffer createNormalBuffer()
    {
        FloatBuffer buffer = ByteBuffer.allocateDirect(_normals.length *
     4).asFloatBuffer();
        buffer.put(_normals);
        buffer.rewind();
        return buffer;
    }

    private FloatBuffer createColorBuffer()
    {
        FloatBuffer buffer = ByteBuffer.allocateDirect(_colors.length *
     4).asFloatBuffer();
        buffer.put(_colors);
        buffer.rewind();
        return buffer;
    }
```

```
    private void createEGLContext()
    {
        // Create an EGL context
        _eglContext = _egl.eglCreateContext
            (_eglDisplay, _eglConfig, EGL10.EGL_NO_CONTEXT, null);

        // Get the GL interface for our new context
        _gl = (GL10)_eglContext.getGL();

        // Make our new context current
        _egl.eglMakeCurrent
           (_eglDisplay, _eglSurface, _eglSurface, _eglContext);
    }

    private void destroyEGL()
    {
        _egl.eglMakeCurrent(_eglDisplay, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_CONTEXT);
        _egl.eglDestroyContext(_eglDisplay, _eglContext);
        _egl.eglDestroySurface(_eglDisplay, _eglSurface);
    }

    private void handleContextLost()
    {
        // Destroy our EGL context
        _egl.eglMakeCurrent(_eglDisplay, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_CONTEXT);
        _egl.eglDestroyContext(_eglDisplay, _eglContext);
        _eglContext = EGL10.EGL_NO_CONTEXT;

        // Re-create our EGL context
        createEGLContext();
    }

    /**
     * Main render loop.
     */
    public void run()
    {
        initialize();
        int throttle = 0;

        while (_running)
        {

            throttle = (int)System.currentTimeMillis();

            // Idle If we are in the background
            if (_paused)
            {
                synchronized (this)
```

```
                {
                    try
                    {
                        wait();
                    }
                    catch (InterruptedException x) { }
                }
            }

            updateBackBuffer();
            renderFrame();
            ++_angle;

            if (_angle >= 360f)
            {
                _angle = 0f;
            }

            //Determine how long the frame took to render.
            throttle = (int)System.currentTimeMillis() - throttle;

            //Throttle to 30 FPS to control CPU usage.
            throttle = 33 - throttle;

            if (throttle > 0)
            {
                // Throttle cpu usage
                try
                {
                    Thread.sleep(throttle);
                }
                catch (InterruptedException x) { }
            }
        }

        destroyEGL();
    }

    private void renderFrame()
    {
        // Make our context and surface current and check for EGL_CONTEXT_LOST
        if (!_egl.eglMakeCurrent(_eglDisplay, _eglSurface, _eglSurface,
     _eglContext))
        {
            if (_egl.eglGetError() == EGL11.EGL_CONTEXT_LOST)
                handleContextLost();
        }

        // Signal that we are about to begin OpenGL rendering
        _egl.eglWaitNative(EGL10.EGL_CORE_NATIVE_ENGINE, _offscreenGraphics);

        render(_gl);
```

```
        // Signal that OpenGL ES rendering is complete
        _egl.eglWaitGL();

        // Swap the window surface to the display
        _egl.eglSwapBuffers(_eglDisplay, _eglSurface);
    }

    private void render(GL10 gl)
    {
        // Set our GL viewport
        gl.glViewport(0, 0, getWidth(), getHeight());

        // Clear the surface
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        net.rim.device.api.opengles.GLUtils.gluPerspective(gl, 45.0f,
     (float)getWidth()/(float)getHeight(), 0.15f, 100.0f);

        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();

        // Setup drawing state
        gl.glEnable(GL10.GL_DEPTH_TEST);
        gl.glEnable(GL10.GL_LIGHTING);
        gl.glEnable(GL10.GL_LIGHT0);
        gl.glEnable(GL10.GL_COLOR_MATERIAL);

        // Draw our cube
        gl.glTranslatef(0, 0, -3.5f);
        gl.glRotatef(_angle, 1.0f, 1.0f, 0.0f);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, _cubeVertices);
        gl.glNormalPointer(GL10.GL_FLOAT, 0, _cubeNormals);
        gl.glColorPointer(4, GL10.GL_FLOAT, 0, _cubeColors);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, _vertexCount);
    }

    /**
     * Called by the UI system to paint the screen.
     */
    protected void paint(Graphics g)
    {
        if (_offscreenBitmap != null)
            g.drawBitmap(0, 0, _offscreenBitmap.getWidth(),
                _offscreenBitmap.getHeight(), _offscreenBitmap, 0, 0);
    }
```

```
    /**
     * Called when the visibility of our screen changes.
     *
     * @param visible true if our screen is being made visible,
     * false if it's being hidden
     */
    protected void onVisibilityChange(boolean visible)
    {
        if (visible)
        {
            resume();
        }
        else
        {
            pause();
        }
    }

    /**
     * Called when the screen is closing.
     */
    public void close()
    {
        _running = false;
        synchronized (this) { notifyAll(); }

        super.close();
    }

    /**
     * Keeps the back buffer in sync with the screen size.
     */
    private void updateBackBuffer()
    {
        if (_offscreenBitmap != null)
        {
            if (_offscreenBitmap.getWidth() == getWidth() &&
                _offscreenBitmap.getHeight() == getHeight())
                return; // no change needed
        }

        _offscreenBitmap = new Bitmap(getWidth(), getHeight());
        _offscreenGraphics = Graphics.create(_offscreenBitmap);
    }

    private void pause()
    {
        _paused = true;
    }

    private void resume()
```

```
    {
        if (_running)
        {
            // Pause the render loop
            _paused = false;
            synchronized (this) { notifyAll(); }
        }
        else
        {
            // Start the render thread.
            _running = true;
            new Thread(this).start();
        }
    }
}
```

Code sample: Drawing a 3-D cube

You can use the `GL10` interface to draw a 3-D cube.

```
import java.nio.ByteBuffer;
import java.nio.FloatBuffer;

import javax.microedition.khronos.egl.EGL10;
import javax.microedition.khronos.egl.EGL11;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.egl.EGLContext;
import javax.microedition.khronos.egl.EGLDisplay;
import javax.microedition.khronos.egl.EGLSurface;
import javax.microedition.khronos.opengles.GL10;

import net.rim.device.api.system.Bitmap;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.container.FullScreen;

class ThreeDCube extends FullScreen implements Runnable
{
    private EGL11 _egl;
    private EGLDisplay _eglDisplay;
    private EGLConfig _eglConfig;
    private EGLSurface _eglSurface;
    private EGLContext _eglContext;
    private GL10 _gl;

    private Bitmap _offscreenBitmap;
    private Graphics _offscreenGraphics;
```

```
    private boolean _running;
    private boolean _paused;
    private FloatBuffer _cubeVertices, _cubeNormals, _cubeColors;

    float _angle = 45f;

    private int _vertexCount;

    private static final float[] _vertices =
    {
        // front
        -0.5f, 0.5f, 0.5f, -0.5f, -0.5f, 0.5f, 0.5f, 0.5f, 0.5f,
        0.5f, 0.5f, 0.5f, -0.5f, -0.5f, 0.5f, 0.5f, -0.5f, 0.5f,

        // right
        0.5f, 0.5f, 0.5f, 0.5f, -0.5f, 0.5f, 0.5f, 0.5f, -0.5f,
        0.5f, 0.5f, -0.5f, 0.5f, -0.5f, 0.5f, 0.5f, -0.5f, -0.5f,

        // back
        0.5f, 0.5f, -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, 0.5f, -0.5f,
        -0.5f, 0.5f, -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, -0.5f, -0.5f,

        // left
        -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, -0.5f, -0.5f, 0.5f, 0.5f,
        -0.5f, 0.5f, 0.5f, -0.5f, -0.5f, -0.5f, -0.5f, -0.5f, 0.5f,

        // top
        -0.5f, 0.5f, -0.5f, -0.5f, 0.5f, 0.5f, 0.5f, 0.5f, -0.5f,
        0.5f, 0.5f, -0.5f, -0.5f, 0.5f, 0.5f, 0.5f, 0.5f, 0.5f,

        // bottom
        -0.5f, -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, 0.5f, -0.5f, 0.5f,
        0.5f, -0.5f, 0.5f, -0.5f, -0.5f, -0.5f, 0.5f, -0.5f, -0.5f
    };

    private static final float[] _normals =
    {
        /* front */ 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
        /* right */ 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
        /* back */ 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1,
        /* left */ -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0,
        /* top */ 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
        /* bottom */ 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0
    };

    private static final float[] _colors =
    {
        /* front  - white  */  1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1,
     1,1,1,1,
        /* right  - red    */  1,0,0,1, 1,0,0,1, 1,0,0,1, 1,0,0,1, 1,0,0,1,
     1,0,0,1,
        /* back   - green  */  0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1,
```

```
     0,1,0,1,
        /* left   - blue   */  0,0,1,1, 0,0,1,1, 0,0,1,1, 0,0,1,1, 0,0,1,1,
     0,0,1,1,
        /* top    - yellow */  1,1,0,1, 1,1,0,1, 1,1,0,1, 1,1,0,1, 1,1,0,1, 1,1,0,1,
        /* bottom - magenta*/  1,0,1,1, 1,0,1,1, 1,0,1,1, 1,0,1,1, 1,0,1,1, 1,0,1,1
    };


    ThreeDCube()
    {
        super(FullScreen.DEFAULT_MENU | FullScreen.DEFAULT_CLOSE);
    }

    private void initialize()
    {
        // Get EGL interface
        _egl = (EGL11)EGLContext.getEGL();

        // Get the EGL display
        _eglDisplay = _egl.eglGetDisplay(EGL11.EGL_DEFAULT_DISPLAY);

        // Initialize the display for EGL, null since we don't really need the
//  version.
        _egl.eglInitialize(_eglDisplay, null);

        // Choose an EGL config
        EGLConfig[] configs = new EGLConfig[1];
        int[] numConfigs = new int[1];
        int[] attrs =
        {
            EGL11.EGL_RED_SIZE,   5,
            EGL11.EGL_GREEN_SIZE, 6,
            EGL11.EGL_BLUE_SIZE,  5,
            EGL11.EGL_NONE
        };
        _egl.eglChooseConfig(_eglDisplay, attrs, configs, 1, numConfigs);
        _eglConfig = configs[0];

        // Create an EGL window surface
        _eglSurface = _egl.eglCreateWindowSurface
            (_eglDisplay, _eglConfig, this, null);

        // Create an EGL context
        createEGLContext();

        _cubeVertices = createVertexBuffer();
        _cubeNormals = createNormalBuffer();
        _cubeColors = createColorBuffer();

        _vertexCount = _vertices.length / 3;
    }
```

```
    private FloatBuffer createVertexBuffer()
    {
        FloatBuffer buffer = ByteBuffer.allocateDirect(_vertices.length *
     4).asFloatBuffer();
        buffer.put(_vertices);
        buffer.rewind();
        return buffer;
    }

    private FloatBuffer createNormalBuffer()
    {
        FloatBuffer buffer = ByteBuffer.allocateDirect(_normals.length *
     4).asFloatBuffer();
        buffer.put(_normals);
        buffer.rewind();
        return buffer;
    }

    private FloatBuffer createColorBuffer()
    {
        FloatBuffer buffer = ByteBuffer.allocateDirect(_colors.length *
     4).asFloatBuffer();
        buffer.put(_colors);
        buffer.rewind();
        return buffer;
    }


    private void createEGLContext()
    {
        // Create an EGL context
        _eglContext = _egl.eglCreateContext
            (_eglDisplay, _eglConfig, EGL10.EGL_NO_CONTEXT, null);

        // Get the GL interface for our new context
        _gl = (GL10)_eglContext.getGL();

        // Make our new context current
        _egl.eglMakeCurrent
           (_eglDisplay, _eglSurface, _eglSurface, _eglContext);
    }

    private void destroyEGL()
    {
        _egl.eglMakeCurrent(_eglDisplay, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_CONTEXT);
        _egl.eglDestroyContext(_eglDisplay, _eglContext);
        _egl.eglDestroySurface(_eglDisplay, _eglSurface);
    }

    private void handleContextLost()
    {
```

```
        // Destroy our EGL context
        _egl.eglMakeCurrent(_eglDisplay, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_CONTEXT);
        _egl.eglDestroyContext(_eglDisplay, _eglContext);
        _eglContext = EGL10.EGL_NO_CONTEXT;

        // Re-create our EGL context
        createEGLContext();
    }

    /**
     * Main render loop.
     */
    public void run()
    {
        initialize();
        int throttle = 0;

        while (_running)
        {

            throttle = (int)System.currentTimeMillis();

            // Idle If we are in the background
            if (_paused)
            {
                synchronized (this)
                {
                    try
                    {
                        wait();
                    }
                    catch (InterruptedException x) { }
                }
            }

            updateBackBuffer();
            renderFrame();
            ++_angle;

            if (_angle >= 360f)
            {
                _angle = 0f;
            }

            //Determine how long the frame took to render.
            throttle = (int)System.currentTimeMillis() - throttle;

            //Throttle to 30 FPS to control CPU usage.
            throttle = 33 - throttle;

            if (throttle > 0)
```

```
            {
                // Throttle cpu usage
                try
                {
                    Thread.sleep(throttle);
                }
                catch (InterruptedException x) { }
            }
        }

        destroyEGL();
    }

    private void renderFrame()
    {
        // Make our context and surface current and check for EGL_CONTEXT_LOST
        if (!_egl.eglMakeCurrent(_eglDisplay, _eglSurface, _eglSurface,
     _eglContext))
        {
            if (_egl.eglGetError() == EGL11.EGL_CONTEXT_LOST)
                handleContextLost();
        }

        // Signal that we are about to begin OpenGL rendering
        _egl.eglWaitNative(EGL10.EGL_CORE_NATIVE_ENGINE, _offscreenGraphics);

        render(_gl);

        // Signal that OpenGL ES rendering is complete
        _egl.eglWaitGL();

        // Swap the window surface to the display
        _egl.eglSwapBuffers(_eglDisplay, _eglSurface);
    }

    private void render(GL10 gl)
    {
        // Set our GL viewport
        gl.glViewport(0, 0, getWidth(), getHeight());

        // Clear the surface
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        net.rim.device.api.opengles.GLUtils.gluPerspective(gl, 45.0f,
     (float)getWidth()/(float)getHeight(), 0.15f, 100.0f);

        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
```

```
        // Setup drawing state
        gl.glEnable(GL10.GL_DEPTH_TEST);
        gl.glEnable(GL10.GL_LIGHTING);
        gl.glEnable(GL10.GL_LIGHT0);
        gl.glEnable(GL10.GL_COLOR_MATERIAL);

        // Draw our cube
        gl.glTranslatef(0, 0, -3.5f);
        gl.glRotatef(_angle, 1.0f, 1.0f, 0.0f);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, _cubeVertices);
        gl.glNormalPointer(GL10.GL_FLOAT, 0, _cubeNormals);
        gl.glColorPointer(4, GL10.GL_FLOAT, 0, _cubeColors);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, _vertexCount);
    }

    /**
     * Called by the UI system to paint the screen.
     */
    protected void paint(Graphics g)
    {
        if (_offscreenBitmap != null)
            g.drawBitmap(0, 0, _offscreenBitmap.getWidth(),
                _offscreenBitmap.getHeight(), _offscreenBitmap, 0, 0);
    }

    /**
     * Called when the visibility of our screen changes.
     *
     * @param visible true if our screen is being made visible,
     * false if it's being hidden
     */
    protected void onVisibilityChange(boolean visible)
    {
        if (visible)
        {
            resume();
        }
        else
        {
            pause();
        }
    }

    /**
     * Called when the screen is closing.
     */
    public void close()
    {
        _running = false;
```

```
        synchronized (this) { notifyAll(); }

        super.close();
    }

    /**
     * Keeps the back buffer in sync with the screen size.
     */
    private void updateBackBuffer()
    {
        if (_offscreenBitmap != null)
        {
            if (_offscreenBitmap.getWidth() == getWidth() &&
                _offscreenBitmap.getHeight() == getHeight())
                return; // no change needed
        }

        _offscreenBitmap = new Bitmap(getWidth(), getHeight());
        _offscreenGraphics = Graphics.create(_offscreenBitmap);
    }

    private void pause()
    {
        _paused = true;
    }

    private void resume()
    {
        if (_running)
        {
            // Pause the render loop
            _paused = false;
            synchronized (this) { notifyAll(); }
        }
        else
        {
            // Start the render thread.
            _running = true;
            new Thread(this).start();
        }
    }
}
```

3-D math utilities

You can use the methods thet are provide in the `net.rim.device.api.math` package to enhance the mathematical versatility and processing speed of BlackBerry® device applications that you create.

Code sample: Using fixed-point arithmetic

The `math.Fixed32` class is a collection of fixed-point arithmetic methods that use the 16.16 convention to fix the decimal point. This conventiom designates the most-significant 16-bits of the 32-bit `int` as the fixed-point component of the value, and the remainder as the decimal component

The following code sample demonstrates how you can and cannot use the `Fixed32` class for addition, subtraction, multiplication, division, value conversion, and trigonometry.

```
import net.rim.device.api.math.*;

public class demoFixed32
{
    demoFixed32()
    {
        // convert an integer to fixed-point
        int n = Fixed32.toFP(7); // 7.0

        // convert a quantity in ten-thousandths to fixed-point
        int m = Fixed32.tenThouToFP(70625); // 7.0625

        // convert from fixed-point, truncate fractional component
        int trunc = Fixed32.toInt(m); // 7

        // multiply by ten thousand
        int mult = Fixed32.toIntTenThou(m); // 70625

        // add, subtract, negate, and compare
        int result = m - n; // 7.0625 - 7.0 == 0.0625
        result = -result; // -0.0625
        result -= n; // -0.0625 - 7.0 == -7.0625
        boolean b = (result == -m); // true
        boolean bb = (m < n); // false

        // do not use increment and decrement operators
        result = Fixed32.toFP(2);
        ++result; // WRONG! result will NOT be 3

        result = Fixed32.toFP(2);
        result += Fixed32.toFP(1); // Correct: result will be 3

        // Use * and / when multiplying or dividing by an integer scalar
        // Use mul to multiply 2 fixed-point numbers
        // Use div to divide 2 fixed-point numbers
        m = Fixed32.tenThouToFP(12500); // 1.25
        m *= 3; // OK: 1.25 * 3 == 3.75
        m /= 2; // OK: 3.75 / 2 == 1.875
        m = Fixed32.mul(m, Fixed32.tenThouToFP(15000)); // 1.875 * 1.5000 == 2.8125
```

```
        m = Fixed32.div(m, m); // 2.8125 / 2.8125 == 1.0

        // mul, div, sqrt, sind, cosd, tand, and atand2
        // all work with 16.16 fixed-point numbers
        m = Fixed32.tenThouToFP(172500); // 17.2500
        n = Fixed32.sqrt(m); // sqrt(17.25)
        n = Fixed32.sind(m); // sine of 17.25 degrees
        n = Fixed32.cosd(m); // cosine of 17.25 degrees
        result = Fixed32.atand2(-m, -m); // -135.0 degrees in fixed-point
    }
}
```

Glossary

9

API
application programming interface

CLDC
Connected Limited Device Configuration

EVDO
Evolution Data Optimized

GSM
Global System for Mobile Communications®

HTTP
Hypertext Transfer Protocol

HTTPS
Hypertext Transfer Protocol over Secure Sockets Layer

I/O
input/output

JSR
Java® Specification Request

JVM
Java® Virtual Machine

MIDP
Mobile Information Device Profile

PCM
pulse code modulation

RAPC
RIM Application Program Compiler

RMS
Record Management System

RTSP
Real Time Streaming Protocol

SVG
Scalable Vector Graphics

TCP
Transmission Control Protocol

TLS
Transport Layer Security

Provide feedback

10

To provide feedback on this deliverable, visit www.blackberry.com/docsfeedback.

Document revision history

11

Date	Description
27 July 2010	Changed the following topics: • Display an image for zooming and panning • Code sample: Displaying an image for zooming and panning • Display a row of images for scrolling • Code sample: Displaying a row of images for scrolling • RIM proprietary video format (RIMM streaming file)
14 August 2009	Added the following topics: • RIMM streaming video file • RIM proprietary video format (RIMM streaming file) • Code sample: Parsing a RIMM streaming video file • Code sample: Drawing a 3-D cube
14 August 2009	Added the following topics: • Displaying an image for zooming and panning • Display an image for zooming and panning • Code sample: Displaying an image for zooming and panning • Displaying a row of images for scrolling • Display a row of images for scrolling • Code sample: Displaying a row of images for scrolling

Legal notice

12

This documentation including all documentation incorporated by reference herein such as documentation provided or made available at www.blackberry.com/go/docs is provided or made accessible "AS IS" and "AS AVAILABLE" and without condition, endorsement, guarantee, representation, or warranty of any kind by Research In Motion Limited and its affiliated companies ("RIM") and RIM assumes no responsibility for any typographical, technical, or other inaccuracies, errors, or omissions in this documentation. In order to protect RIM proprietary and confidential information and/or trade secrets, this documentation may describe some aspects of RIM technology in generalized terms. RIM reserves the right to periodically change information that is contained in this documentation; however, RIM makes no commitment to provide any such changes, updates, enhancements, or other additions to this documentation to you in a timely manner or at all.

This documentation might contain references to third-party sources of information, hardware or software, products or services including components and content such as content protected by copyright and/or third-party web sites (collectively the "Third Party Products and Services"). RIM does not control, and is not responsible for, any Third Party Products and Services including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third Party Products and Services. The inclusion of a reference to Third Party Products and Services in this documentation does not imply endorsement by RIM of the Third Party Products and Services or the third party in any way.

EXCEPT TO THE EXTENT SPECIFICALLY PROHIBITED BY APPLICABLE LAW IN YOUR JURISDICTION, ALL CONDITIONS, ENDORSEMENTS, GUARANTEES, REPRESENTATIONS, OR WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY CONDITIONS, ENDORSEMENTS, GUARANTEES, REPRESENTATIONS OR WARRANTIES OF DURABILITY, FITNESS FOR A PARTICULAR PURPOSE OR USE, MERCHANTABILITY, MERCHANTABLE QUALITY, NON-INFRINGEMENT, SATISFACTORY QUALITY, OR TITLE, OR ARISING FROM A STATUTE OR CUSTOM OR A COURSE OF DEALING OR USAGE OF TRADE, OR RELATED TO THE DOCUMENTATION OR ITS USE, OR PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE, HARDWARE, SERVICE, OR ANY THIRD PARTY PRODUCTS AND SERVICES REFERENCED HEREIN, ARE HEREBY EXCLUDED. YOU MAY ALSO HAVE OTHER RIGHTS THAT VARY BY STATE OR PROVINCE. SOME JURISDICTIONS MAY NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES AND CONDITIONS. TO THE EXTENT PERMITTED BY LAW, ANY IMPLIED WARRANTIES OR CONDITIONS RELATING TO THE DOCUMENTATION TO THE EXTENT THEY CANNOT BE EXCLUDED AS SET OUT ABOVE, BUT CAN BE LIMITED, ARE HEREBY LIMITED TO NINETY (90) DAYS FROM THE DATE YOU FIRST ACQUIRED THE DOCUMENTATION OR THE ITEM THAT IS THE SUBJECT OF THE CLAIM.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, IN NO EVENT SHALL RIM BE LIABLE FOR ANY TYPE OF DAMAGES RELATED TO THIS DOCUMENTATION OR ITS USE, OR PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE, HARDWARE, SERVICE, OR ANY THIRD PARTY PRODUCTS AND SERVICES REFERENCED HEREIN INCLUDING WITHOUT LIMITATION ANY OF THE FOLLOWING DAMAGES: DIRECT, CONSEQUENTIAL, EXEMPLARY, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR AGGRAVATED DAMAGES, DAMAGES FOR LOSS OF PROFITS OR REVENUES, FAILURE TO REALIZE ANY EXPECTED SAVINGS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF

BUSINESS OPPORTUNITY, OR CORRUPTION OR LOSS OF DATA, FAILURES TO TRANSMIT OR RECEIVE ANY DATA, PROBLEMS ASSOCIATED WITH ANY APPLICATIONS USED IN CONJUNCTION WITH RIM PRODUCTS OR SERVICES, DOWNTIME COSTS, LOSS OF THE USE OF RIM PRODUCTS OR SERVICES OR ANY PORTION THEREOF OR OF ANY AIRTIME SERVICES, COST OF SUBSTITUTE GOODS, COSTS OF COVER, FACILITIES OR SERVICES, COST OF CAPITAL, OR OTHER SIMILAR PECUNIARY LOSSES, WHETHER OR NOT SUCH DAMAGES WERE FORESEEN OR UNFORESEEN, AND EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, RIM SHALL HAVE NO OTHER OBLIGATION, DUTY, OR LIABILITY WHATSOEVER IN CONTRACT, TORT, OR OTHERWISE TO YOU INCLUDING ANY LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY.

THE LIMITATIONS, EXCLUSIONS, AND DISCLAIMERS HEREIN SHALL APPLY: (A) IRRESPECTIVE OF THE NATURE OF THE CAUSE OF ACTION, DEMAND, OR ACTION BY YOU INCLUDING BUT NOT LIMITED TO BREACH OF CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR ANY OTHER LEGAL THEORY AND SHALL SURVIVE A FUNDAMENTAL BREACH OR BREACHES OR THE FAILURE OF THE ESSENTIAL PURPOSE OF THIS AGREEMENT OR OF ANY REMEDY CONTAINED HEREIN; AND (B) TO RIM AND ITS AFFILIATED COMPANIES, THEIR SUCCESSORS, ASSIGNS, AGENTS, SUPPLIERS (INCLUDING AIRTIME SERVICE PROVIDERS), AUTHORIZED RIM DISTRIBUTORS (ALSO INCLUDING AIRTIME SERVICE PROVIDERS) AND THEIR RESPECTIVE DIRECTORS, EMPLOYEES, AND INDEPENDENT CONTRACTORS.

IN ADDITION TO THE LIMITATIONS AND EXCLUSIONS SET OUT ABOVE, IN NO EVENT SHALL ANY DIRECTOR, EMPLOYEE, AGENT, DISTRIBUTOR, SUPPLIER, INDEPENDENT CONTRACTOR OF RIM OR ANY AFFILIATES OF RIM HAVE ANY LIABILITY ARISING FROM OR RELATED TO THE DOCUMENTATION.

Prior to subscribing for, installing, or using any Third Party Products and Services, it is your responsibility to ensure that your airtime service provider has agreed to support all of their features. Some airtime service providers might not offer Internet browsing functionality with a subscription to the BlackBerry® Internet Service. Check with your service provider for availability, roaming arrangements, service plans and features. Installation or use of Third Party Products and Services with RIM's products and services may require one or more patent, trademark, copyright, or other licenses in order to avoid infringement or violation of third party rights. You are solely responsible for determining whether to use Third Party Products and Services and if any third party licenses are required to do so. If required you are responsible for acquiring them. You should not install or use Third Party Products and Services until all necessary licenses have been acquired. Any Third Party Products and Services that are provided with RIM's products and services are provided as a convenience to you and are provided "AS IS" with no express or implied conditions, endorsements, guarantees, representations, or warranties of any kind by RIM and RIM assumes no liability whatsoever, in relation thereto. Your use of Third Party Products and Services shall be governed by and subject to you agreeing to the terms of separate licenses and other agreements applicable thereto with third parties, except to the extent expressly covered by a license or other agreement with RIM.

Certain features outlined in this documentation require a minimum version of BlackBerry® Enterprise Server, BlackBerry® Desktop Software, and/or BlackBerry® Device Software.

The terms of use of any RIM product or service are set out in a separate license or other agreement with RIM applicable thereto. NOTHING IN THIS DOCUMENTATION IS INTENDED TO SUPERSEDE ANY EXPRESS WRITTEN AGREEMENTS OR WARRANTIES PROVIDED BY RIM FOR PORTIONS OF ANY RIM PRODUCT OR SERVICE OTHER THAN THIS DOCUMENTATION.

Research In Motion Limited
295 Phillip Street

Waterloo, ON N2L 3W8
Canada

Research In Motion UK Limited
Centrum House
36 Station Road
Egham, Surrey TW20 9LF
United Kingdom

Published in Canada